TUGAS AKHIR – TE 141599

METODE PENCACAHAN FREKUENSI RECIPROCAL UNTUK SENSOR GAS RESONATOR KUARSA YANG DIIMPLEMENTASIKAN PADA *FIELD PROGRAMMABLE GATE ARRAY*

Reza Barkah Harjunadi
NRP 2209100035

Dosen Pembimbing
Dr. Muhammad Rivai, ST., MT.
Rudy Dikairono, ST., MT.

JURUSAN TEKNIK ELEKTRO
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya 2015

FINAL PROJECT – TE 141599

FREQUENCY COUNTER RECIPROCAL METHOD FOR QUARTZ RESONATOR GAS SENSOR IMPLEMENTED ON A FIELD PROGRAMMABLE GATE ARRAY

Reza Barkah Harjunadi
NRP 2209100035

Supervisor
Dr. Muhammad Rivai, ST., MT.
Rudy Dikairono, ST., MT.

ELECTRICAL ENGINEERING DEPARTMENT
Faculty of Industrial Technology
Institute Technology of Sepuluh Nopember
Surabaya 2015

METODE PENCACAHAN FREKUENSI RECIPROCAL UNTUK SENSOR GAS RESONATOR KUARSA YANG DIIMPLEMENTASIKAN PADA *FIELD PROGRAMMABLE GATE ARRAY*

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Elektronika
Jurusan Teknik Elektro
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember

Menyetujui :

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Muhammad Rivai, ST., MT.	**Rudy Dikairono, ST., MT.**
NIP. 196904261994031003	**NIP. 19810325200501002**

SURABAYA
JANUARI, 2015

METODE PENCACAHAN FREKUENSI RECIPROCAL UNTUK SENSOR GAS RESONATOR KUARSA YANG DIIMPLEMENTASIKAN PADA *FIELD PROGRAMMABLE GATE ARRAY*

Reza Barkah Harjunadi
2209100035

Dosen Pembimbing 1 : Dr. Muhammad Rivai, ST., MT.
Dosen Pembimbing 2 : Rudy Dikairono, ST., MT.

ABSTRAK

Quartz Crystal Microbalance (QCM) merupakan salah satu jenis resonator kuarsa yang memiliki membran sensitif terhadap gas. Pada sistem identifikasi gas menggunakan QCM, perubahan frekuensi dari sensor ini begitu cepat, sehingga untuk mendapatkan perubahan frekuensinya diperlukan metode pencacahan yang lebih cepat dibanding metode yang biasanya digunakan. Beberapa aplikasi QCM digunakan pada sistem identifikasi gas. Untuk mendapatkan luaran QCM diperlukan sebuah proses instrumentasi, salah satu cara yang pernah dirancang adalah menggunakan sistem pencacah *reciprocal frequency*. Sistem ini berbasis digital yang tersusun atas rangkaian diferensial frekuensi, rangkaian pembagi frekuensi, dan rangkaian pencacah. Pada penelitian ini sistem digital direalisasikan menggunakan *Field Programmable Gate Array* (FPGA). FPGA memiliki kelebihan diantaranya jenis dan jumlah gerbangnya sangat banyak, dan mudah diprogram berkali-kali. Pada perancangan sistem ini untuk mendapatkan pergeseran frekuensi dilakukan dengan membandingkan antara frekuensi referensi (fr) dan frekuensi probe sensor (fx). Hasilnya berupa selisih frekuensi (fd) yang dibagi menggunakan rangkaian pembagi frekuensi. Pencacahan dilakukan dengan menggunakan frekuensi referensi (fr) dan selisih frekuensi yang di bagi (fd/N) sebagai periodenya. Data dari pencacahan ini dikirim ke komputer untuk proses identifikasi menggunakan *neural network*. Prosentase keberhasilan keseluruhan sistem dalam mengidentifikasi gas uji sebesar 80%.

Kata kunci : FPGA, *Quartz Crystal Microbalance*, *Reciprocal frequency counter*, Resonator Kuarsa.

FREQUENCY COUNTER RECIPROCAL METHOD FOR QUARTZ RESONATOR GAS SENSOR IMPLEMENTED ON A FIELD PROGRAMMABLE GATE ARRAY

Reza Barkah Harjunadi
2209100035

1st Advisor : Dr. Muhammad Rivai, ST., MT.
2nd Advisor : Rudy Dikairono, ST., MT.

ABSTRACT

Quartz Crystal Microbalance (QCM) is a type of quartz resonator which has a gas-sensitive membrane. In the gas identification system using QCM, the changing of the sensor's frequency is very fast, in order to obtain the frequency changes counting method that is faster than the previous method is required. One of the applications which used QCM is gas identification. In order to get the QCM's output, instrumentation process is required, one way ever designed is using a system of reciprocal frequency counter. This digital-based system is composed of a differential frequency circuit, dividing frequency circuit, and the counter circuit. On this research, the digital system will be implemented using Field Programmable Gate Array (FPGA). FPGA has advantages such as its many type and number of Gates, and is easily programmed many times. On the design of this system, obtaining a frequency shift is done by comparing the frequency of reference (fr) and the frequency of the sensor probe (fx). The result is the difference frequency (fd) which are divided using a frequency divider. Counting is carried out using the frequency of reference (fr) and the difference frequency of (fd/N) as a periode. Data from this counting is sent to a computer to process the identification using neural network. Percentage of the overall success of the system in identifying the tested gas by 80 %.

Keywords : FPGA, Quartz Crystal Microbalance, Quartz Resonator, Reciprocal frequency counter*.*

KATA PENGANTAR

Alhamdulillah, segala puji dan rasa sykur penulis panjatkan kehadirat Allah SWT sehingga penulis dapat menyelesaikan tugas akhir ini.

Tugas Akhir ini disusun sebagai salah satu persyaratan untuk memperoleh gelar sarjana pada Jurusan Teknik Elektro, Fakultas Teknologi Industri, Institut Teknologi Sepuluh Nopember Surabaya.

Selama pelaksanaan penelitian Tugas Akhir ini, penulis mendapatkan bantuan dari berbagai pihak, dan penulis sampaikan rasa terima kasih. Terima kasih yang sebesar-besarnya juga penulis sampaikan kepada berbagai pihak yang mendukung dan membantu dalam tugas akhir ini.

Untuk itu pada kesempatan ini penulis menyampaikan ucapan terimakasih kepada:

1. Keluarga yang telah memberikan dukungan dan motivasi serta doa untuk menyelesaikan tugas akhir ini.
2. Bapak Dr. Muhammad Rivai, ST., MT. selaku dosen pembimbing pertama, atas bimbingan, inspirasi, pengarahan, dan motivasi yang diberikan selama pengerjaan penelitian tugas akhir ini.
3. Bapak Rudy Dikairono, ST., MT. selaku dosen pembimbing kedua, atas bimbingan, inspirasi, pengarahan, dan motivasi yang diberikan selama pengerjaan penelitian tugas akhir ini.
4. Bapak Ir. Tasripan, MT., selaku koordinator Bidang studi Elektronika.
5. Dosen bidang studi Elektronika dan juga seluruh dosen Jurusan Teknik Elektro yang banyak memberikan ilmu selama penulis menempuh kuliah.
6. Rekan-rekan yang banyak membantu dan memberi semangat serta dukungan dan doa dalam penyelesaian tugas akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari kesempurnaan dan masih banyak hal yang perlu diperbaiki. Saran, kritik dan masukan baik dari semua pihak sangat membantu penulis terutama untuk berbagai kemungkinan pengembangan lebih lanjut. Harapan penulis adalah semoga apa yang telah ditulis dalam buku ini dapat

bermanfaat bagi para pembaca sekalian, serta agar tugas akhir ini dapat dikembangkan lagi menjadi lebih baik. Amin,

Surabaya, 22 Januari 2014

Reza Barkah Harjunadi

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1. Latar Belakang

Proses pencacahan frekuensi pada umumnya memerlukan waktu selama satu detik. Metode ini disebut dengan metode *conventional frequency counter*. Pada tahun 80-an ditemukan sebuah metode yang dapat mencacah frekuensi selama periode yang diinginkan. Metode ini disebut dengan *reciprocal frequency counter*.

Salah satu sensor gas resonator kuarsa yang dapat digunakan untuk mengukur suatu massa yang sangat kecil dengan data luaran berupa frekuensi adalah *Quartz Crystal Microbalance* (QCM). QCM biasa digunakan pada pendeteksian gas. QCM memiliki kelebihan-kelebihan dibandingkan dengan sensor gas lainnya. Sensor ini mempunyai sensitivitas yang tinggai dan stabil dalam suhu yang tinggi [1].

QCM digunakan untuk menerima gas yang ditiupkan dan akan menghasilkan perubahan frekuensi yang berbeda untuk tiap gas yang berbeda. Untuk mendapatkan luaran QCM diperlukan sebuah proses instrumentasi, salah satu cara yang pernah dirancang adalah menggunakan sistem pencacah frekuensi [2].

Sistem ini berbasis digital yang tersusun atas rangkaian diferensial frekuensi, rangkaian pembagi frekuensi, dan rangkaian pencacah. Agar frekuensi yang dihasilkan oleh sensor dapat dihitung, digunakanlah teknologi *Field Programmable Gate Array* (FPGA) yang memiliki banyak keunggulan dibanding mikrokontroler. FPGA dapat digunakan untuk pengimplementasian rangkaian logika yang sangat kompleks pada chip tunggal tanpa harus mendesain rangkaian secara kasar yang menghabiskan banyak waktu. Apabila terdapat kesalahan, proses desain pada FPGA dapat diulang secara cepat tanpa harus mengubah perangkat kerasnya. Terlebih lagi bila ingin menambah jumlah sensor yang akan digunakan, kita hanya perlu memodifikasi program pada FPGA.

Pada tugas akhir ini dirancang sistem *reciprocal frequency counter* yang menggunakan FPGA sebagai pengganti sistem digitalnya. Pada FPGA akan dibuat beberapa blok utama antara lain pemilih frekuensi sensor, *frequency difference*, pembagi frekuensi, pencacah frekuensi, dan blok untuk komunikasi serial dengan PC.

Kemudian data hasil penghitungan frekuensi akan diproses dengan menggunakan software yang menggunakan algoritma jaringan saraf tiruan pada PC untuk mengenali jenis uap yang ditiupkan pada sensor.

1.2. Perumusan Masalah

Permasalahan yang dibahas dalam tugas akhir ini adalah:

1. Bagaimana merancang sistem *reciprocal frequency counter* untuk sensor gas QCM pada FPGA.
2. Bagaimana proses pengolahan data sehingga gas dapat diidentifikasi dengan cepat dan akurat.

1.3. Tujuan Penelitian

Penelitian pada tugas akhir ini bertujuan sebagai berikut :

1. Mampu merancang dan mengimplementasikan *reciprocal frequency counter* untuk sensor gas.
2. Mampu merancang dan mengimplementasikan sistem pembanding frekuensi pada FPGA.
3. Mampu merancang dan mengimplementasikan sistem pembagi frekuensi pada FPGA.
4. Mampu memahami proses akuisisi data dengan menggunakan sensor QCM yang berpolimer.
5. Mampu memproses data sehingga pergeseran frekuensi dapat diketahui secara akurat.

1.4. Batasan Masalah

Batasan masalah dari tugas akhir ini adalah sebagai berikut :

1. Sensor yang digunakan berupa *Quartz Crystal Microbalance* yang diberi polimer yang ditentukan.
2. Pengujian alat dilakukan pada kondisi suhu dan kelembapan yang sama
3. Proses pencacahan frekuensi untuk sensor gas dilakukan dengan menggunakan FPGA yang dikomunikasikan dengan PC.

1.5. Metodologi Penelitian

Dalam penyelesaian tugas akhir ini digunakan metodologi sebagai berikut :

1. Studi literatur

a. Studi tentang karakteristik QCM

b. Studi tentang karekteristik polimer yang digunakan pada QCM
c. Studi tentang *reciprocal frequency counter*
d. Studi tentang FPGA Spartan-3e dan akuisisi data yang dibutuhkan

2. **Perancangan Hardware**
   a. Perancangan sensor deret QCM
   b. Pembuatan rangkaian osilator
3. **Perancangan Software**
   Pada tahap ini dirancang sebuah sofrware yang diprogram pada FPGA Spartan-3e. Program ini terdiri dari selektor, rangkaian logika pembanding frekuensi, rangkaian logika pembagi frekuensi, counter, dan paralel to serial yang akan dilanjutkan ke PC.
4. **Pengujian sistem**
   Proses pengujian sistem dilakukan dengan menggunaan gas-gas tertentu. Kemudian dianalisa hasil yang telah didapatkan. Pengujian ini dilakukan untuk mengetahui tingkat keberhasilan dari sistem yang telah dirancang.
5. **Penulisan laporan Tugas Akhir**
   Tahap penulisan laporan Tugas Akhir dilakukan pada saat tahap pengujian sistem dimulai serta setelahnya. Penulisan laporan menunjukkan hasil akhir dari tugas akhir.

1.6. Sistematika Penulisan

Laporan tugas akhir ini terdiri dari lima bab dengan sistematika penulisan sebagai berikut:

- Bab 1 : Pendahuluan
  Bab ini meliputi latar belakang, perumusan masalah, tujuan penelitian, batasan masalah, metodologi, sistematika penulisan, dan relevansi.
- Bab 2 : Dasar Teori
  Bab ini menjelaskan tentang teori dasar yang dibutuhkan dalam pengerjaan tugas akhir ini, yang meliputi arsitektur kuarsa resonator, cara kerja sensor, dan rangkaian oscillator dan *neural network*.
  Bab 3: Perancangan Alat

Bab ini menjelaskan tentang perencanaan sistem baik perangkat keras (hardware) maupun perangkat lunak (software) untuk pengidentifikasian gas uji

- Bab 4 : Pengujian Alat
  Bab ini menjelaskan tentang hasil yang didapat dari tiap blok sistem dan subsistem yang telah dirancang dan analisa data dari hasil perhitungan yang dilakukan.
- Bab 5 : Penutup
  Bab ini menjelaskan tentang kesimpulan meliputi kekurangan-kekurangan pada kerja alat dari hasil analisa serta saran untuk pengembangan ke depan.

1.7. Relevansi

Hasil dari tugas akhir ini diharapkan dapat memberikan manfaat sebagai berikut :

1. Dapat mendukung penelitian khususnya pada bidang studi elektronika tentang penggunaan sensor gas untuk aplikasi pengukuran dan penelitian selanjutnya.
2. Dihasilkan alat pendeteksi kadar-kadar kandungan dalam gas yang ke depannya dapat diaplikasikan langsung dalam dunia industri.

BAB II
DASAR TEORI

2.1. Reciprocal Frequency Counter

Frequency counter adalah pencacahan banyaknya siklus frekuensi per lama waktu yang digunakan dalam mencacah, dimana hal ini sesuai pada persamaan berikut

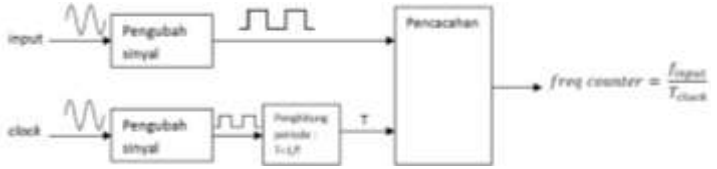
$$frekuensi\ (Hz) = \frac{banyak\ siklus}{waktu\ pengukuran}$$

Dari persamaan diatas dapat disimpulkan bahwa setidaknya membutuhkan dua sinyal untuk melakukan pencacahan.

Metode pertama yang digunakan dalam pencacahan frekuensi ini adalah metode *Conventional*. Metode ini menghitung banyaknya siklus frekuensi per satu detik. Hal ini tidak sesuai dengan persamaan diatas, dimana waktu pengukurannya sudah ditentukan. Setidaknya pada metode ini dibutuhkan frekuensi dengan besar 1Hz yang digunakan sebagai lama waktu pengukuran.

Pada tahun 1980-an munculah metode *Reciprocal*, dimana metode ini menerapkan persamaan pencacahan frekuensi diatas. Pada metode ini membutuhkan dua register pencacah, register pertama untuk mencacah banyaknya siklus input, dan yang kedua untuk mencacah banyaknya pulsa *clock,* untuk mengukur durasi waktu. Dua gerbang utama disinkronkan mengontrol register pencacah secara simultan.

Berbeda dengan metode *Conventional*, pada metode ini waktu pengukuran didapatkan melalui besar periode yang dibutuhkan untuk satu siklus *clock*. Sedangkan pada metode *Conventional*, waktu pengukuran sudah ditentukan yaitu selama satu detik [1].



Gambar 2.1. Diagram blok *Reciprocal Frequency Counter*

Resolusi dari pencacahan frekuensi metode *Conventional* dibatasi hanya dapat menghitung siklus lengkap dari frekuensi sinyal input.

Pengukuran akan dimulai pada fase acak dari sinyal input dan akan berhenti secara acak pula disuatu tempat selama siklus dari frekuensi sinyal input. Tidak ada cara untuk dapat mengetahui berapa banyak siklus yang terlewat pada proses pencacahan, ini bisa melewatkan hampir satu siklus lengkap. Oleh karena itu resolusi untuk satu detik pengukuran adalah ±1Hz.

Pada metode *Reciprocal*, untuk mendapatkan resolusi pencacahan yang tinggi dapat mengubah besar frekuensi sinyal *clock*. Semakin tinggi frekuensi *clock*, maka waktu penghitungan akan semakin cepat serta resolusi penghitungan akan semakin kurang akurat, sebaliknya bila semakin rendah frekuensi *clock*, waktu penghitungan akan semakin lama, resolusi penghitungan akan semakin akurat [2].

Metode ini memiliki keunggulan salah satunya adalah ±1 error perhitungan tidak bergantung pada frekuensi sinyal input. Oleh karena itu untuk sinyal input ber-*noise* dan dengan mengabaikan *trigger* dan error basis waktu, resolusi dari metode *Reciprocal* juga akan bergantung pada sinyal input [3].

2.2. Resonator Kuarsa

Pada tahun 1959, Sauerbrey memperkenalkan resonator kuarsa pertama kali. Dia menunjukkan penemuan tentang karakteristik kristal kuarsa. Kristal ini memiliki sifat yang dapat menghasilkan tegangan listrik ketika diberi tegangan mekanikal dan juga sebaliknya, berubah bentuk mekanikalnya ketika diberi tegangan listrik. Sifat ini disebut dengan sifat *piezoelectric* [4].

Dengan adanya sifat *piezoelectric*, maka resonator kuarsa dapat digunakan sebagai sensor kimiawi. Resonator kuarsa yang memiliki membran yang sensitif terhadap gas dapat digunakan untuk pendeteksian gas gas. Molekul gas yang ditiupkan dan menempel pada membran yang sensitif ini akan menyebabkan perubahan pada massa membran yang akan mengakibatkan perubahan pada resonan frekuensi dari frekuensi awal. Namun ketika melokul gas ini telah lepas dari membran, frekuensi resonator ini akan kembali pada frekuensi awalnya. Hal ini biasa disebut dengan nama *mass loading effect* [5].

Perubahan frekuensi yang terjadi pada resonator kuarsa ini, dirumuskan oleh Sauerbrey sebagai berikut:

$$\Delta f = -(\frac{2f^2}{\rho v A})\Delta m$$

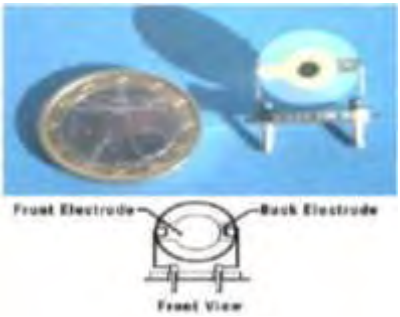
Keterangan:

Δf adalah perubahan frekuensi yang diamati (Hz).
f adalah frekuensi resonansi dasar (Hz).
Δm adalah perubahan massa per unit area (kg).
ρ adalah kerapatan kristal (kg/m^3).
v adalah kecepatan propagasi akustik pada kristal (m/s).
A adalah luas elektroda (m^2).

Dari persaaman diatas dapat disimpulkan bahwa dengan menggunakan kuarsa resonator yang telah ditentukan, besar dari kerpatan kristal (ρ), kecepatan propagasi akustik kristal (v), dan luas elektroda (A) tidak akan berubah. Semakin besar perubahan massa (Δm) akibat menempelnya molekul gas, akan mengakibatkan perubahan frekuensi (Δf) semakin rendah. Semakin besar nilai frekuensi resonansi dasar (f) maka perbuahan frekuensinya (Δf) akan semakin besar [6].

2.2.1 Quartz Crystal Microbalance (QCM)

Quartz Crystal Microbalance (QCM) adalah salah satu resonator kuarsa yang memiliki membran yang sensitif yang dapat digunakan sebagai sensor elektronik serta mempunyai fungsi seperti indera penciuman. Pengembangan QCM yang digunakan untuk gas cairan membuka dunia baru dalam hal aplikasi, termasuk elektrokimia [7].

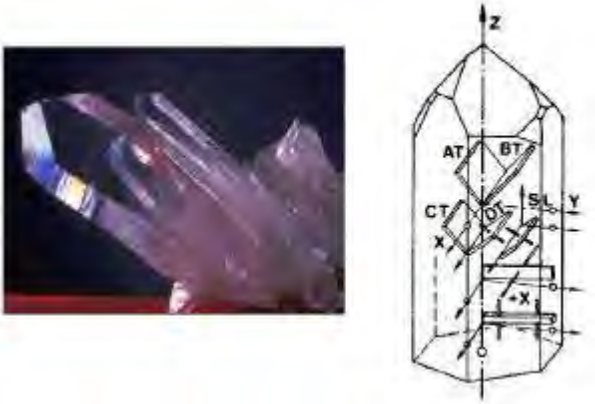


Gambar 2.2. Sensor QCM [6]

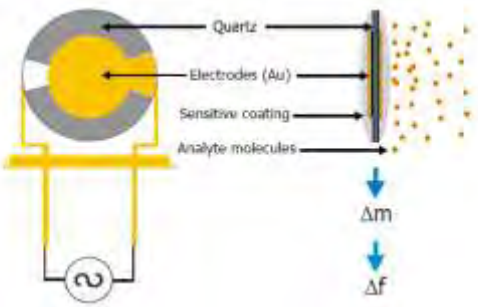
Frekuensi resonansi dasar pada QCM ditentukan oleh potongan dan dimensi keping kristal yang digunakan pada saat pembuatan. Sehinggal struktur dari sensor QCM adalah kristal

yang terdapat lapisan SiO_2 dan diapit oleh dua elektrode sehingga dapat menghasilkan potensial listrik sebagai respon terhadap tekanan mekanik yang diberikan [8].

QCM juga dikenal bekerja sebagai TSM (*Thickness Sheer Mode*), ini adalah sejenis tipe BAW (*Bulk Acoustic Wave*), tipe sensor massa yang sangat sensitif, dan secara tradisional digunakan untuk kontrol pendeposisian lapisan tipis. Sudut pemotongan terhadap orientasi kristal berpengaruh terhadap besarnya nilai osilasi. Ada banyak standar potongan keping kristal, yang masing-masing memiliki karakteristik yang berbeda-beda. Contohnya potongan AT yang populer memiliki frekuensi fundamental maksimum yang tidak terlalu tinggi dan koefisien suhu yang cukup baik (berbentuk kurva fungsi kubik). Lainnya potongan BT yang memiliki frekuensi fundamental maksimum yang lebih baik, namun koefisien suhunya lebih buruk (berbentuk kurva parabola). Elektrode kristal yang digunakan untuk penerapan aplikasi QCM biasanya terbuat dari emas (Au). Selain Au juga ada lainnya, misalnya Cu, Cr, Ni, Pt logam.



Gambar 2.3. Kristal kuarsa alami (kiri) dan ilustrasi kristal kuarsa dengan perbedaan sudut potong (kanan) [9]



Gambar 2.4. Prinsip QCM [9]

Dalam praktiknya, sensor QCM terdiri dari disk tipis, yang diiris dari kristal tunggal dari sebuah kuarsa tipe *alpha*. Kristal ini berada diantara dua elektroda logam dimana uap disimpan di kedua sisi kristal. Sensitivitas frekuensi osilasi kristal pada suhu, kelembaban, tekanan, kecepatan dan getaran tertentu akan membuat osilator *piezoelectric* ini digunakan sebagai sensor yang mempunyai tingkat akurasi yang tinggi [9].

2.2.2 Polimer QCM

Sebuah metode yang digunakan saat ini melibatkan penggunaan deret (array) sensor yang mencontoh dari sistem penciuman biologi dimana reseptor penciumannya tidak berselektif tinggi terhadap gas tertentu. Apabila hanya menggunakan sebuah reseptor untuk menanggapi uap tertentu, hasilnya akan kurang memuaskan. Dengan menggunakan reseptor yang lebih dari satu maka akan mendapatkan pola tanggapan yang berbeda dari setiap elemen sensor, sehingga uap dapat diklasifikasikan. Pada saat ini para peneliti sedang menggunakan deret sensor polimer yang memiliki nilai kepolaran berbeda yang dikombinasi dengan bahan konduksi maupun semikonduksi untuk meniru sistem penciuman. Bahan kristal SiO_2 telah lama digunakan pada peralatan elektronika sebagai pengatur frekuensi. Pada tugas akhir ini, suatu deret kristal terlapis plimer. Bahan polimer yang digunakan mempunyai polaritas yang berbeda berdasar pada ketetapan McReynoldnya, yaitu PEG1540, OV-275, Cellulose, Ap-L. [10].

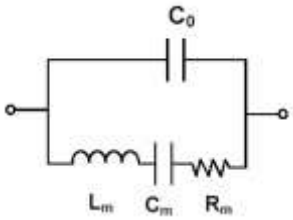
2.2.3 Rangkaian Ekivalen

Rangkaian ekivalen yang dapat diaplikasikan dalam resonator kuarsa adalah model *Butterworth van Dyke* (BVD). Model ini sering digunakan untuk menampilkan *electrical behavior* dari sebuah kristal resonator. Model ini juga baik untuk memprediksi pergeseran frekuensi dan rugi-rugi pada AT-cut kristal kuarsa yang teraplikasi dalam resonator kuarsa. Model *Butterworth van Dyke* terlihat seperti gambar 2.5.

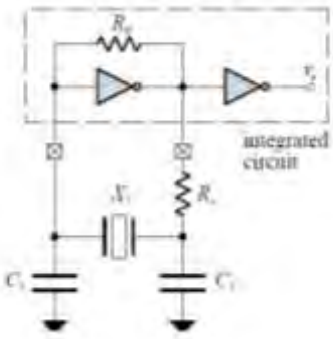
Model BVD ini terdiri dari dua lengan. *Motion arm* mempunyai tiga komponen seri yang termodifikasi dari massa dan *viscous loading crystal*:

1. R_m (resistor) mewakili disipasi energi osilasi dari *mounting* struktur dan dari medium yang terhubung dengan kristal.
2. C_m (kapasitor) mewakili energi osilasi yang tersimpan dan terkait dengan elastisitas kuarsa dan medium sekelilingnya.
3. L_m (induktor) mewakili komponen inersia saat osilasi yang terkait dengan pergerakan massa selama bergetar.

Untuk diameter 1 inchi, 5 MHz kristal, menggunakan nilai C_m=33pF, L_m=30mH dan R_m=10 Ohm (kristal kering), R_m=400 Ohm (kristal dengan sebagian di air) atau R_m=3500 Ohm (kristal dengan bagian 88% di gliserol). *Motional arm* di shunt dengan kapasitor parasitik (Co) yang mana merupakan jumlah dari kapasitansi statis pada elektrode-elektrode kristal, penjepit dan konektor kapasitor. Harga Co sekitar 20 pF [11].



Gambar 2.5. Model rangkaian osilator *Butterworth van Dyke* (BVD) [11]



Gambar 2.6. Rangkaian osilator *pierce* [12]

Dalam tugas akhir ini sensor gas yang digunakan adalah jenis resonator kristal, sehingga dibutuhkan rangkain osilator untuk menjamin sinyal yang dihasilkan resonator kristal adalah sinyal pulsa. Rangkain osilator yang banyak digunakan dalam aplikasi aplikasi digital adalah jenis osilator *pierce*. Rangkaiannya dapat dilihat pada gambar 2.6.

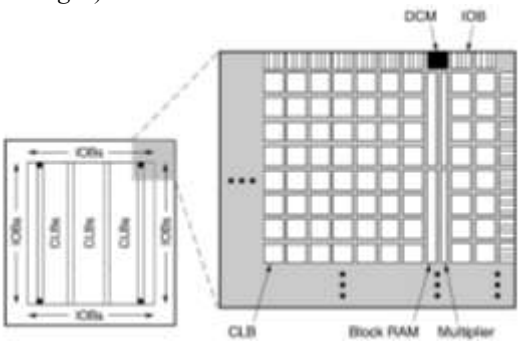
Keunggulan menggunakan Rangkaian osilator ini adalah dapat diimplementasikan menggunakan rangkaian yang minimum, dengan dua digital inverter, dua resistor, dua kapasitor, dan sebuah kristal kuarsa. Dengan harga yang relatif rendah dan kestabilan frekuensi dari kristal kuarsa dibandingkan dengan desain rangkaian osilator yang lain, banyak para teknisi elektronika menggunakannya. Untuk frekuensi kristal sebesar 20Mhz diperlukan R_b sebesar 1MΩ, sedangkan untuk frekuensi diatas itu diperlukan R_b yang lebih tinggi lagi nilainya. Umumnya para desainer elektronika menggunakan nilai C_1 dan C_2 sama sebesar 20pf [12].

2.3. Field Programmable Gate Array (FPGA)

FPGA memiliki ribuan bahkan lebih transistor yang saling terintegrasi di dalam satu chip ini. Dalam proses menciptakan sebuah IC, FPGA mengintegrasikan transistor-transistor yang ada di dalamnya. Sehingga FPGA termasuk dalam teknologi VLSI (*Very Large Scale Integration*). VLSI ini sendiri sudah berkembang sejak FPGA yang pertama [13].

Field Programmable Gate Array (FPGA) merupakan sebuah IC yang terdiri dari blok blok logika yang interkoneksinya dapat diprogram. Struktur elektrikal konfigurasi yang terinterkoneksi memperbolehkan aplikasi rangkaian yang kompleks untuk di buat pada beberapa sel yang tersedia dalam FPGA. Saat ini FPGA muncul dalam beberapa seri keluarga dimana hanya dibedakan dengan banyaknya sel. Harga yang relatif murah dan daya rendah chip FPGA dimulai dari beberapa ribu sel, pada harga pertengahan dan daya rata-rata memiliki sekitar sepuluh ribu sel, puluhan 18-bit pengali secara paralel dan 2kB blok memori, dan chip FPGA yang termahal memiliki seratus ribuan sel, 2000 pengali, serta beberapa *Mbytes* memori. Dalam chip FPGA mengandung ribuan bahkan lebih transistor [14].

Secara umum FPGA terdiri dari *logic block* dan blok input/ouput. Semua blok didalamnya dapat diprogram sampai pada skala tertentu. Untuk generasi FPGA terbaru, memiliki kemampuan yang tinggi dalam proses logika dengan konfigurasi mencapai 550MHz, dibandingkan dengan mikrokontroler yang hanya mampu sampai 24Mhz. Dengan kecepatan tersebut dimungkinkan dalam perancangan sistem sistem yang lebih kompleks seperti, pengolahan citra, *Digital Signal Processing*, pengukuran frekuensi tinggi memakai teknologi FPGA. Secara umum arsitektur FPGA Spartan-III terlihat dari gambar 2.7, terdiri dari atas 5 komponen utama, CLB(*Configurable Logic Block*), IOB(*Input/Output Block*), *Multiplier Block*, BR(*Block* RAM), DCM (*Digital Clock Manager*).



Gambar 2.7. Arsitektur keluarga Spartan-3 [15]

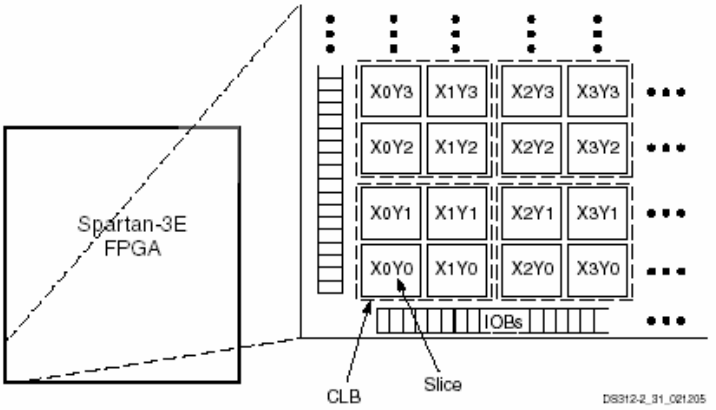
Arsitektur keluarga Spartan-3 terdiri dari lima komponen utama pemrograman dasar elemen fungsi, CLB(*Configurable Logic Block*),

IOB(*Input/Output Block*), *Multiplier Block*, BR(*Block* RAM), DCM (*Digital Clock Manager*) [15].

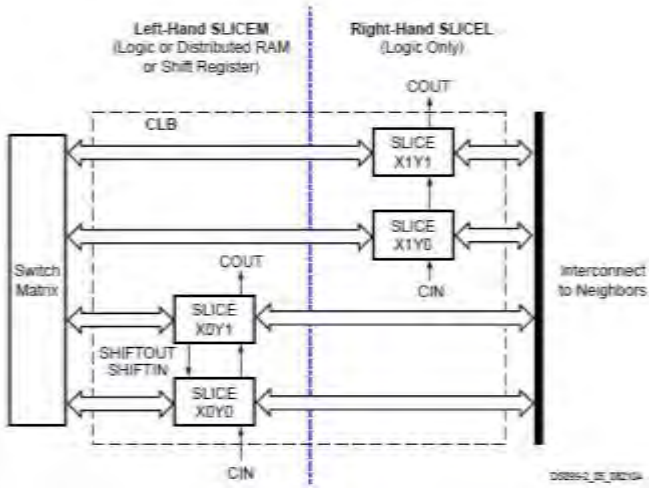
2.3.1 Configurable Logic Block (CLB)

Configurable Logic Block (CLB) merupakan bagian utama logika yang mengimplementasi sinkronisasi dan rangkaian kombinasional. Setiap CLB mengandung empat *slice*, dan setiap *slice* mengandung dua *Look-Up Tables* (LUT) untuk mengimplementasikan logika dan dua penyimpanan elemen khusus yang dapat digunakan sebagai flip-flop atau penguncian. CLB tersusun pada baris dan kolom seperti terlihat pada gambar 2.8.

Setiap *slice* yang berada didalam CLB dikelompokkan menjadi pasangan, setiap pasangan disusun sebagai kolom dengan rantai *carry* independen. Pasangan dikiri mensuport kedua logika dan fungsi memori *slice* ini disebut SLICEM. Pasangan dikanan hanya mensuport logika dan *slice* ini disebut SLICEL. Interkoneksi dari SLICEM dan SLICEL dengan matrik *switch* serta komponen yang lain dapat dilihat pada gambar 2.9.



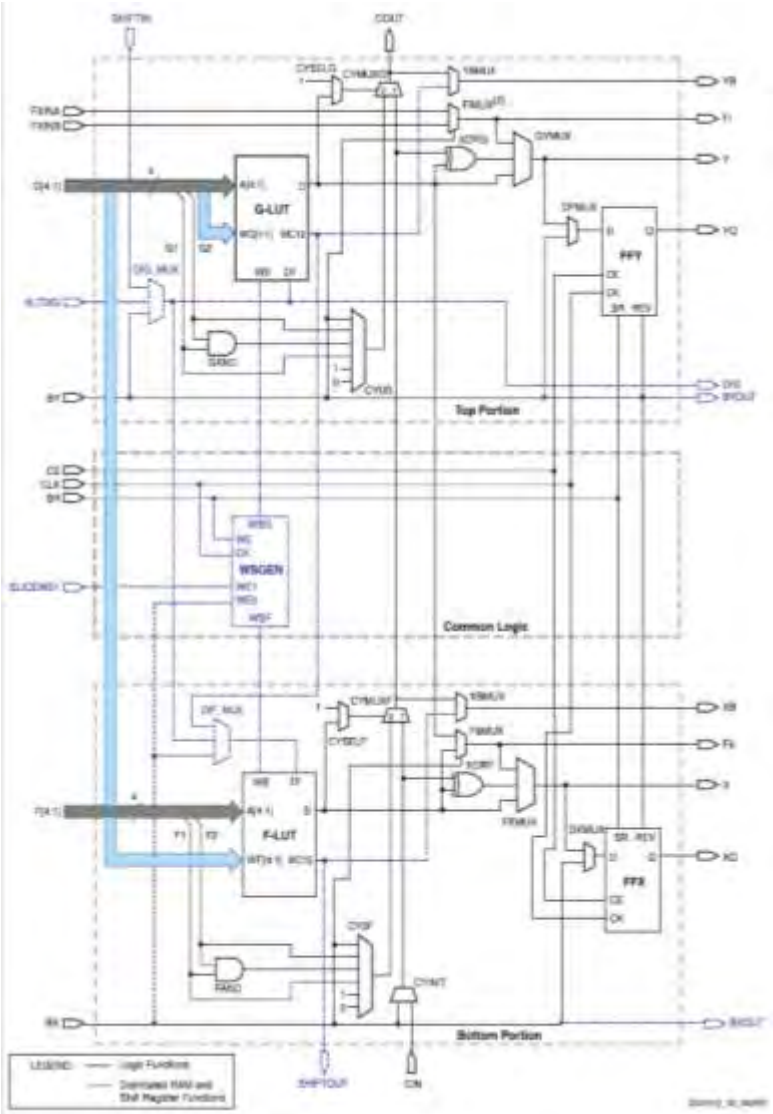
Gambar 2.8. Lokasi CLB [16]



Gambar 2.9. Interkoneksi SLICEM dan SLICEL dengan matrik *switch* dan komponen lain [16]

Untuk setiap bagian-bagian dari CLB, memiliki penjelasan sebagai berikut:

1. LUT (*Look-Up Table*), setiap LUT memiliki 4 buah input dan dapat dikonfigurasikan sebagai *function generator* atau sebagai 16x1 bit SRAM. Kedua LUT dalam *slice* dapat digabungkan untuk membentuk 16x2 bit SRAM, 32x1 bit SRAM, ataupun 16x1 bit *dual-port* SRAM. LUT juga dapat dikonfigurasikan menjadi 16-bit *shift register* yang dapat dioperasikan dengan *clock* kecepatan tinggi.
2. *Multiplexer*, setiap multiplexer F5 digunakan untuk memilih output dari 2 *function generator* pada *slice*, sedangkan *multiplexer* F6 digunakan untuk memilih output dari *multiplexer* F5.
3. *Storage Elements*, elemen ini dapat dikonfigurasikan sebagai edge-triggered D-Flip_flop atau sebagai *level-sensitive latch*. Setiap elemen dapat dikemudikan melalui *output* dari *function generator* ataupun langsung dari *input slice* dengan mem-*bypass function generator*.



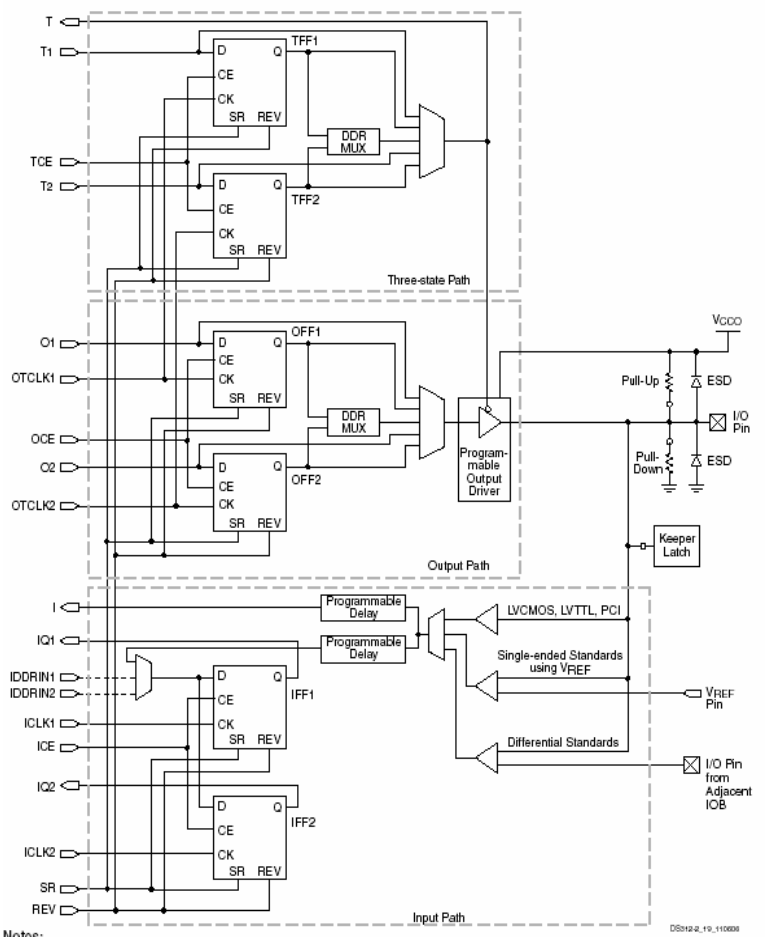
Gambar 2.10. Diagram sederhana dari SLICEM [16]

4. *Arithmatic Logic*, tersusun atas *dedicated carry logic*, XOR *gate* dan AND *gate*. *Dedicated carry logic* memungkinkan untuk perhitungan aritmatika kecepatan tinggi. Kedua gerbang memungkinkan untuk dibuatnya 1-bit *full adder* dalam satu LC dan juga meningkatkan efisiensi dari proses perkalian. *Dedicated carry path* juga dapat digunakan untuk mengkaskade *function generator* untuk mengimplementasikan fungsi logika yang besar.
5. *Three State Buffer* (BUFT). Setiap BUFT memiliki control dan input tersendiri digunakan untuk mengemudikan bus internal [11].

2.3.2 Input Output Block (IOB)

Input/Output Block (IOB) menyediakan pemrograman, searah atau dua arah *interface* antara pin paket dan logika internal pada FPGA, mensuport berbagai *interface* standard. Gambar 2.11 adalah diagram sederhana struktur internal dari IOB. Terdapat tiga jalur sinyal utama dalam IOB: jalur *output,* jalur *input*, dan jalur 3-*state*. Setiap jalur memiliki pasangan elemen penyimpannya yang dapat berfungsi sebagai pe-*register* atau pe-*latch*.

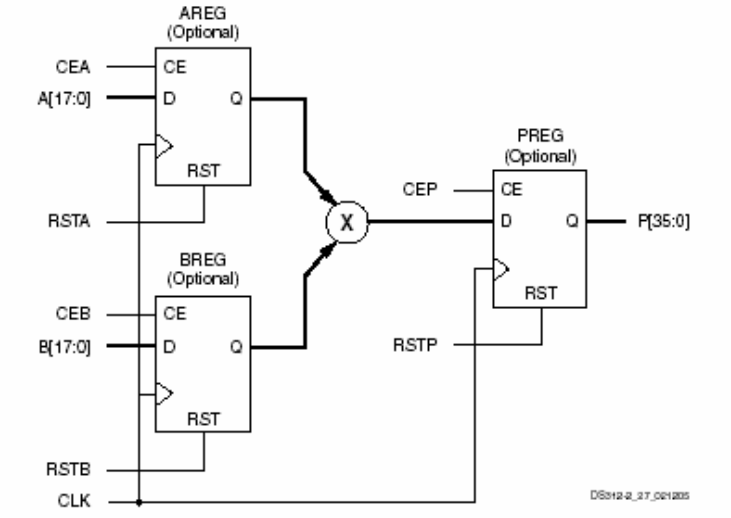
1. Jalur *input*, membawa data dari *pad*, yang menyatu dengan paket pin, melewati pemrograman elemen *delay* menuju jalur I. Setelah elemen *delay*, terdapat rute alternatif melewati sepasang elemen penyimpan menuju jalur IQ1 dan IQ2.
2. Jalur *output*, dimulai dengan jalur O1 dan O2, membawa data dari logika internal FPGA melewati *multiplexer* lalu driver 3-*state* ke *pad* IOB. Selain jalur *direct* ini, *multiplexer* menyediakan pilihan untuk memasukkan sepasang elemen penyimpanan.
3. Jalur 3-*state* menentukan ketika impedansi *output* tinggi. Jalur T1 dan T2 membawa data dari logika internal FPGA melewati multiplexer ke driver *output*. Sama dengan halnya pada jalur *output*, selain jalur *direct* ini, *multiplexer* menyediakan pilihan untuk memasukkan sepasang elemen penyimpanan.



Gambar 2.11. Diagram sederhana IOB [16]

2.3.3 Dedicated Multiplier

Semua keluarga Spartan-3 menyediakan *multipliers* tertanam yang hanya memperbolehkan dua 18-bit kata sebagai *input* untuk memproduksi 36-bit produk.



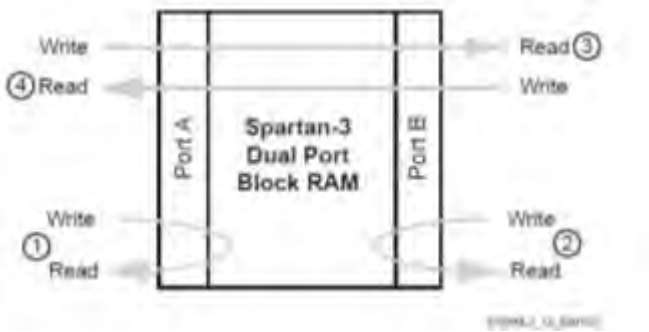
Gambar 2.12. Blok *dedicated multiplier* [11]

Input bus untuk multiplier, menerima data dalam bentuk *2's complement* (baik 18-bit yang terdaftar atau 17-bit yang belum terdaftar) [16].

Dari gambar 2.13 menunjukkan prinsip operasi, P = AxB, dimana 'A' dan 'B' adalah 18-bit kata dalam bentuk *2's complement*, dan 'P' adalah hasil perkalian 36-bit, juga dalam bentuk *2's complement*. Inputan 18-bit mewakili nilai antara -131.071 sampai +131.071 dengan hasil perkalian antara -17.179.738.112 sampai +17.179.738.112 [11].

2.3.4 Block RAM (BR)

Semua keluarga Spartan-3 mendukung BR, dimana diorganisasikan sebagimana terkonfigurasi, mensinkronkan 18Kbit blok. BR menyimpan data dalam jumlah yang relatif besar. BR memiliki struktur dual port. Dua port data yang identik disebut A dan B memperbolehkan akses independen ke common BR, yang mana memiliki kapasitas maksimum 18.432 bit atau 16.8384 bit ketika tidak ada jalur *parity* yang digunakan.



Gambar 2.13. Jalur data Blok RAM [16]

Tiap port memiliki tetapan data, kontrol dan jalur *clock* untuk mensinkronisasi operasi *read* dan *write* [15].

Pada keluarga Spartan-3, BR memiliki kapasitas besar yang memungkinkan untuk diperbolehkannya struktur memori yang lebih besar dari yang disediakan oleh LUT pada CLB [11].

Pada gambar2.13, jalur 1 adalah data input dan output, jalur 2 adalah *parity* input dan output, digunakan ketika port data *byte* lebar atau lebih lebar, jalur 3 adalah alamat input untuk memilih lokasi memori secara spesifik, jalur 4 adalah berbagai sinyal kontrol yang mengelola operasi *read*, *write*, serta *set/reset*.

Kedua port (port A dan B) mengakses banyaknya bit memori yang sama, tetapi dengan dua skema alamat yang berbeda tergantung pada lebar data port.

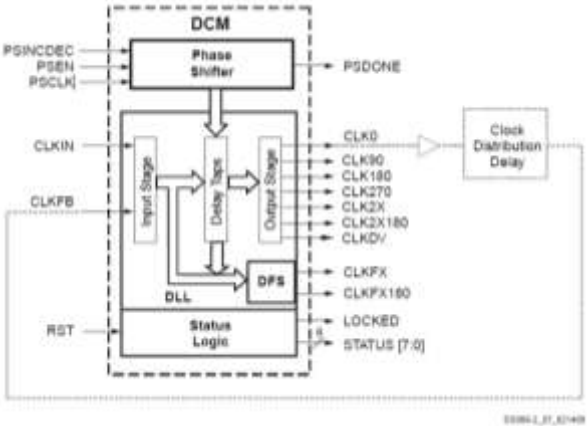
1. Port A berperan sebagai *single-port* RAM independen yang mensuport operasi *read* dan *write* secara simultan menggunakan satu set baris alamat.
2. Port B berperan sebagai *single-port* RAM independen yang mensuport operasi *read* dan *write* secara simultan menggunakan satu set baris alamat.
3. Port A adalah port *write* dengan alamat *write* terpisah, dan port B adalah port *read* dengan alamat *read* terpisah. Lebar data untuk port A dan port B dapat berbeda juga.
4. Port B adalah port *write* dengan alamat *write* terpisah, dan port A adalah port *read* dengan alamat *read* terpisah. Lebar data untuk port B dan port A dapat berbeda juga [16].

2.3.5 Digital Clock Manager (DCM)

DCM mengintegrasikan kemampuan *advance*d clocking secara langsung ke dalam global clock yang terdistribusi di FPGA.

Oleh karena itu DCM-lah yang biasanya memecahkan berbagai masalah pada *common* clocking, terutama dalam performansi tinggi, aplikasi frekuensi tinggi:

1. Menghilangkan clock yang tidak simetris, baik dalam perangkat atau ke komponen eksternal, untuk meningkatkan kerja sistem secara keseluruhan dan untuk menghilangkan *delay* distribusi clock.
2. Pergeseran fase sinyal clock, baik oleh sebagian periode clock atau dengan penambahan jumlah.
3. Pengali atau pembagi frekuensi clock yang akan datang atau mensistesis frekuensi baru secara lengkap dengan penggabungan pengali dan pembagi clock.
4. *Mirror, forward,* atau *rebuffer* sinyal clock, sering mensismetriskan dan mengubah sinyal clock yang masuk ke I/O standar yang berbeda, contohnya meneruskan dan menngubah LVTTL (*Low Voltage TTL*) ke LVDS (*Low Voltage Differential Signaling*) yang masuk.
5. Salah satu atau semua fungsi diatas.



Gambar 2.14. Blok diagram fungsional DCM [15]

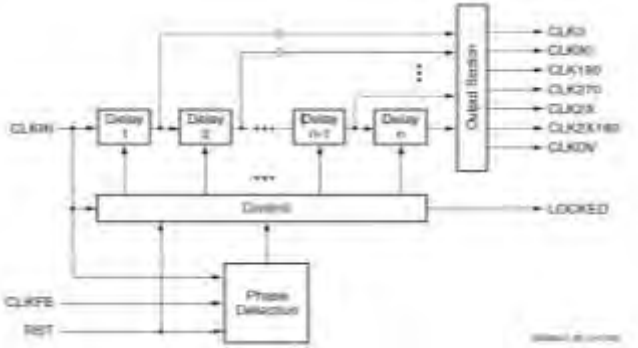
DCM memiliki empat komponen fungsional, antara lain: *Delay-Looked Loop* (DLL), *Digital Frequency Synthesizer* (DFS), *Phase Shifter* (PS), dan status logika komponen. Setiap komponen memiliki sinyal yang terkait.

2.3.5.1. Delay-Locked Loop (DLL)

Fungsi yang paling mendasar dari komponen DLL adalah untuk menghilangkan ketidak simetrisan clock. Jalur utama sinyal DLL terdiri dari inputan, diikuti dengan serangkaian elemen *delay* diskrit, yang mengarah ke output. Jalur ini bersamaan dengan logika untuk mendeteksi fase dan mengontrol pembentukan sistem lengkap dengan *feedback* seperti yang ditunjukkan pada gambar 2.15.

2.3.5.2. Digital Frequency Synthesizer (DFS)

DFS menyediakan berbagai lebar dan fleksibel frekuensi output berdasarkan pada rasio dua tetapan pengguna bilangan bulat. Pengali (CLFX_MULTIPLY) dan pembagi (CLFX_DIVIDE). Frekuensi output berasal dari dari clock input (CLKIN) oleh frekuensi pembagian dan perkalian secara simultan. Fitur DFS ini dapat digunakan dalam hubungannya dengan atau secara terpisah dari fitur DLL di DCM.



Gambar 2.15. Diagram fungsional sederhana DLL [15]

Bila DLL tidak digunakan, output clock DFS tidak akan simetris karena DLL diharuskan memberikan kesimetrisan dari input clock. Unit DFS menghasilkan output *Frequency Synthesizer* (CLKFX, CLKFX180).

2.3.5.3. Phase Shifter (PS)

Unit PS mengontrol fase relasi dari ouput clock DCM menuju input CLKIN.

Unit PS menggeser fase dari kesembilan sinyal clock output DCM dengan fraksi tetap dari perioda clock input. Nilai pergeseran fase tetap diset pada saat perancangan dan dimuat ke DCM selama konfigurasi FPGA. Bila DLL tidak digunakan, clock output DFS tidak akan simetris karena DLL dibutuhkan untuk memberikan kesimetrisan *feedback* clock [16].

2.3.5.4. Status Logika Komponen

Status logika komponen tidak hanya memberi tahu pada keadaan DCM, tetapi juga memberikan sarana *resetting* DCM ke keadaan awal. DCM *reset* (RST) tidak mempengaruhi nilai atribut (misalnya, CLKFX_MULTIPLY dan CLKFX_DIVIDE). Jika tidak digunakan, RST harus dihubungkan ke *ground* (GND) [15].

2.4. Artificial Neural Network (ANN)

Artificial Neural Network (ANN) adalah salah satu representasi buatan dari otak manusia yang selalu mencoba untuk mensimulasikan proses pembelajaran pada otak manusia tersebut. Istilah buatan digunakan karena *neural network* atau lebih dikenal dengan jaringan syaraf ini diimplementasikan dengan menggunakan program komputer yang mampu menyelesaikan sejumlah proses perhitungan selama proses pembelajaran [17].

ANN menyediakan metode umum, praktis untuk belajar nilai nyata, nilai diskrit, dan fungsi nilai vektor dari contoh-contoh. ANN belajar di *error* pada data *train* dan telah berhasil diterapkan untuk masalah-masalah seperti menafsirkan adegan visual, pengenalan suara, dan *learning* strategi pengendalian robot [18].

Menurut Haykin, *Artificial neural network* adalah sejumlah besar prosesor yang terdistribusi secara pararel dan terdiri dari unit pemrosesan sederhana, dimana masing-masing unit memiliki kecenderungan untuk menyimpan pengetahuan yang dialami dan dapat digunakan kembali [19].

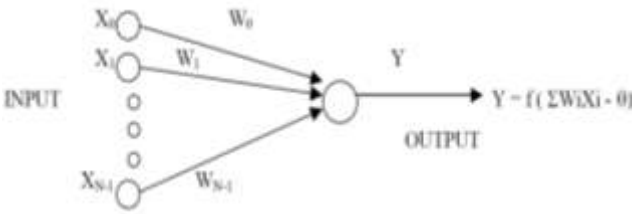
Neural network adalah pertemuan saling berhubungan dari elemen pemrosesan sederhana, unit atau node, yang fungsi secara bebas didasarkan pada neuron hewan. Pengolahan kemampuan jaringan itu disimpan dalam antar-satuan kekuatan sambungan, atau bobot, diperoleh dengan proses adaptasi, atau belajar dari, satu set pelatihan pola.

2.4.1 Struktur Neural Network

Struktur sederhana dari *neural network* dapat dilihat pada gambar 2.16.

Menurut Dozkocs, Reggia, dan Lin, *neural network* memiliki tiga komponen, yaitu *network* atau jaringan, *rule* aktifasi, *rule* pembelajaran.

1. Jaringan mengandung node yang terkoneksi langsung, setiap node di jaringan memiliki nilai aktifasi yang terikat pada waktu t. Keseluruhan pola vektor aktifasi menunjukkan keadaan jaringan saat ini pada waktu t.
2. Aktifasi *rule* adalah prosedur lokal dimana setiap node mengikuti proses pembaharuan aktifasi level masukan dari node tetangga.
3. *Rule* pembelajaran adalah prosedur lokal yang menjelaskan bagaimana berat pada koneksi harus diubah sebagai fungsi waktu [20].



Gambar 2.16. *Neural Network* Sederhana [20]

2.4.2 Mengaktifkan Artificial Neural Network

Banyak fungsi yang dapat digunakan sebagai pengaktif, seperti fungsi-fungsi gonometri dan hiperboliknya, fungsi *unit step*, *impuls*, *sigmoid*. Pada umumnya fungsi yang digunakan adalah fungsi *sigmoid* karena dianggap lebih mendekati kinerja sinyal pada otak manusia [11].

Setiap neuron mempunyai keadaan internal yang disebut level aktivasi atau level aktivitas yang merupakan fungsi input yang diterima. Secara tipikal suatu neuron mengirimkan aktivitasnya kebeberapa neuron lain sebagai sinyal. Yang perlu diperhatikan adalah bahwa neuron hanya dapat mengirimkan satu sinyal sesaat, walaupun sinyal tersebut dapat dipancarkan ke beberapa neuron yang lain.

Ada beberapa pilihan fungsi aktivasi yang digunakan dalam metode *backpropagation*, seperti fungsi sigmoid biner, dan sigmoid bipolar. Karakteristik yang harus dimiliki fungsi aktivasi tersebut adalah kontinyu, diferensiabel, dan tidak menurun secara monoton. Fungsi aktivasi diharapkan dapat mendekati nilai-nilai maksimum dan minimum secara baik. Fungsi aktifasi yang sering digunakan yaitu, fungsi sigmoid unipolar dan sigmoid bipolar.

Fungsi sigmoid unipolar biasa digunakan untuk *neural network* yang dilatih dengan metode *backpropagation*. Biasa digunakan untuk *neural network* yang membutuhkan ouput antara 0 sampai 1. Definisi fungsi sigmoid unipolar adalah sebagai berikut:

$$f_1(x) = \frac{1}{(1 + e^{-x})}$$

dengan turunan :

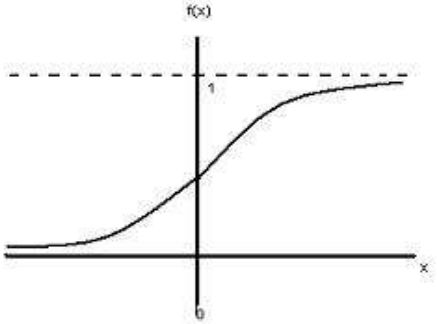
$$f_1'(x) = f_1(x)(1 - f_1(x))$$

Fungsi bipolar sama dengan fungsi unipolar, tetapi output dari fungsi ini antara -1 sampai 1. Definisi fungsi sigmoid bipolar adalah sebagai berikut:

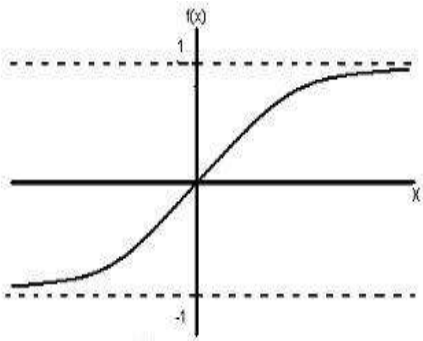
$$f_2(x) = 2\, f_2(x) - 1$$

dengan turunan :

$$f_2'(x) = \frac{1}{2}\big(1 + f_2(x)\big)(1 - f_2(x))$$



Gambar 2.17. Fungsi Sigmoid Unipolar [21]



Gambar 2.18. Fungsi Sigmoid Bipolar [21]

2.4.3 Backpropagation Neural Network

Neural network backpropagation digunakan untuk menentukan besarnya nilai *weight* dan *threshold* setiap layer. Selama proses *learning*, *weight* dan *threshold* diatur sedemikian rupa, untuk memperoleh *error* yang minimum. *Error* (kesalahan) dihitung berdasarkan rata-rata kuadrat kesalahan (MSE). Prinsip dasar dari algoritma *backpropagation* adalah *memperbaiki* bobot-bobot jaringan dengan arah yang membuat fungsi aktivasi menjadi turun dengan cepat [21].

Pelatihan *backpropagation* terdapat 3 tahapan, antara lain tahapan propagasi maju, propagasi mundur, perubahan bobot.

Tahapan propagasi maju, dihitung mulai dari layer input hingga layer output, biasanya menggunakan fungsi aktifasi unipolar. Setiap input dikalikan dengan *weight* dan dijumlahkan dengan *threshold*

$$Z\_in_j = Vo_j + \sum_{i=1}^{n} X_i V_{ij}$$

Kemudian dihitung dengan fungsi aktifasi unipolar

$$Z_j = f(Z\_in_j)$$

Output dari fungsi aktifasi dikirimkan disemua unit di output layer. Tiap output layer dikalikan dengan *weight* dan dijumlahkan dengan *threshold* serta dihitung lagi fungsi aktifasinya. Proses yang sama dilakukan hingga layer output. *Error* didapatkan melalui selisih antara keluaran jaringan dengan target yang diinginkan

Error yang didapatkan tadi digunakan untuk tahapan selanjutnya yaitu propagasi mundur.

Masing-masing *output layer* menerima pola target sesuai dengan pola masukan saat *training* dan dihitung *error*nya :

$$\delta_k = (t_k - y_k) f'(y\_in_k)$$

Karena $f'(y_{in_k}) = y_k$menggunakan fungsi sigmoid, maka :

$$f'(y\_in_k) = f(y_{in_k})(1 - f(y\_in_k))$$
$$= y_k(1 - y_k)$$

Menghitung perbaikan *weight* (kemudian untuk memperbaiki W_{jk})

$$\Delta W_{jk} = \alpha . \partial_k . Z_j$$

Menghitung perbaikan koreksi :

$$\Delta Wo_k = \alpha . \partial_k$$

Dan menggunakan nilai delta (d_k) pada semua *layer* sebelumnya.Masing-masing *weight* yang menghubungkan *output layer* dengan *hidden layer* dikalikan delta (d_k) dan dijumlahkan sebagai masukan untuk *layer* berikutnya :

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k W_{jk}$$

Selanjutnya dikalikan dengan turunan fungsi aktifasi untuk menghitung *error*.

$$\delta_j = \delta\_in_j \, f'(y\_in_j)$$

Langkah berikutnya menghitung perbaikan *weight* :

$$\Delta V_{ij} = \alpha \delta_j X_i$$

Kemudian menghitung perbaikan *threshold* :

$$\Delta Vo_j = \alpha \delta_j$$

Kedua tahapan dilakukan berulang-ulang terus hingga kondisi terpenuhi. Masing-masing output layer diperbaiki *threshold* dan *weight*-nya

$$W_{jk}(baru) = W_{jk}(lama) + \Delta W_{jk}$$

Masing-masing *hidden layer* diperbaiki *threshold* dan *weight*-nya

$$V_{jk}(baru) = V_{jk}(lama) + \Delta V_{jk}$$

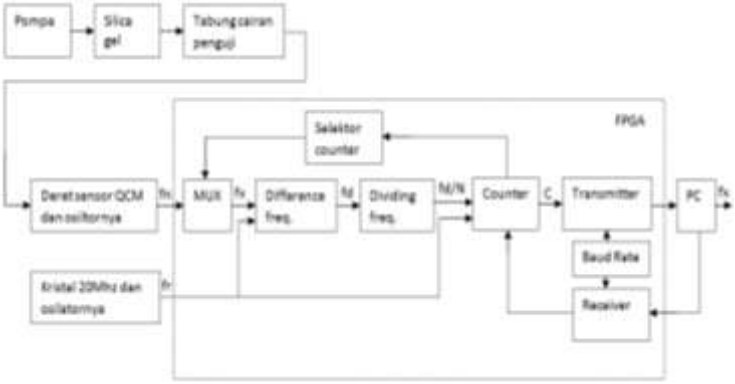
Keterangan :

Xi : nilai aktifasi dari unit Xi.
Zj : unit ke-j pada *hidden layer.*
Z_in_j : output untuk unit Zj.
Yk : unit ke-k pada *output layer.*
Y_in_k : *output* untuk unit Y.
yk : nilai aktifasi dari unit Yk.
Wo_k : nilai *weight* pada *threshold* untuk unit Yk.
Wkj : nilai *weight* dari Zij ke unit Yk.
ΔWkj: selisih antara Wkj (t) dengan Wkj (t+1).
Vjo : nilai *weight* pada *threshold* untuk unit Zj.
Vij : nilai *weight* dari unit Xi ke unit Zj.
ΔVij: selisih antara Vij (t) dengan Vij (t+1).
δk : faktor pengendalian nilai *weight* pada *output layer.*
δj : faktor pengendalian nilai *weight* pada *hidden layer.*
α : konstanta laju *training.*

BAB III
PERANCANGAN SISTEM

Perancangan dan pembuatan alat meliputi perancangan *hardware* dan perancangan *software*. Perancangan *hardware* yang dibuat terdiri atas mendesain sensor QCM dan rangkaian resonator kristal. Kemudian rangkaian *multiplexer*, DFF (sebagai rangkaian pembanding frekuensi), rangkaian pengali frekuensi (*dividing frequency circuit*), pencacah frekuensi, dan *interfacing serial* didesain menggunakan *board* FPGA Spartan 3E *starter kit* dari *digilent inc*. Sedangkan *software* meliputi *software* Xilinx ISE Design Suite 13.2 yang digunakan untuk desain VHDL, sintesis serta mengunduh rancangan ke FPGA serta software untuk mengimplementasikan *Neural Network*.

Pada pembuatan blok diagram sistem seperti pada gambar 3.1, rangkaian osilator menggunakan model osilator pierce. Dalam pengiriman data 24 bit ke komputer memakai komunikasi serial 8 bit, dilakukan pemecahan data sebesar 5 byte, 3 byte data dan 2 byte kode. Sehingga dibutuhkan blok Parsing, untuk memecah datanya menghasilkan 1 byte data yang berisi 8 bit yang dikirim secara bergantian selama lima kali pengiriman.



Gambar 3.1. Blok diagram sistem

Pendeteksian gas dilakukan pada suhu dan kelembaban yang sama, oleh karena itu dirancang mekanik untuk sensor yang dapat

memenuhi kebutuhan tersebut. Pada gambar 3.2 menunjukkan mekanik sensor yang akan digunakan pada tugas akhir ini.

Pemberian gas pada sensor dilakukan menggunakan pompa udara yang diatur dan terhubung dengan mekanik sensor. Sebelum dilakukan pemberian gas pada sensor, gas yang ditiupkan dilewatkan *silica gel* terlebih dahulu.

Gambar 3.2. Mekanik sensor QCM

Gambar 3.3. Gambar pompa udara di hubungkan dengan *silica gel*

Modul FPGA yang digunakan pada tugas akhir ini adalah Spartan-3E dengan chip XC3S250E-4TQ144. Memiliki sistem gerbang 250 ribu, 612 total CLB dan 2.448 *slice*. Untuk suply dari FPGA ini

sendiri hanya manggunakan suply dari port USB. Dibanding FPGA pendahulunya, modul ini lebih berukuran minimalis.

Untuk pengunduhan programnya kedalam FPGA, dibutuhkan sebuah *software*, bitloadapp.

Gambar 3.4. Board FPGA Spartan 3E

Gambar 3.5. Software pengunduh program ke dalam FPGA

Pada *software* bitloadapp ini, perlu tipe file berekstensi bit untuk proses *write* ke dalam FPGA.

Perancangan perangkat lunak pada FPGA menggunakan bahasa VHDL untuk setiap blok pada FPGA. Untuk merealisasikan bahasa

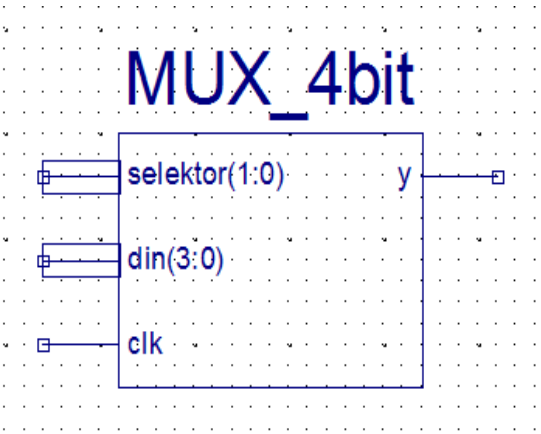
VHDL tiap blok diagram pada modul FPGA, digunakan *software* Xilinx ISE Design Suite 13.2, sebagai pesintesis dan pembuatan bit file.

Terdapat blok diagram utama dalam pembuatan perangkat lunak, antara lain MUX (*multiplexer*), DFF (sebagai *frequency difference*), *Dividing Frequency circuit* (sebagai pembagi frekuensi), dan *counter* (untuk mencacah frekuensi dengan metode *reciprocal frequency counter*).

MUX (*multiplexer*) digunakan sebagai pemilih, agar pembacaan dapat dilakukan secara bergantian.

Pin selektor(1:0) (2bit) sebagai input digunakan untuk memilih pin input (din(3:0)) (4bit) yang akan di keluarkan di output (y). Pin clk digunakan sebagai clock.

DFF (sebagai *frequency difference*), dilakukan proses pengurangan untuk mengurangi interferensi antar sensor yang menyebabkan *noise* pada masing-masing pembacaan frekuensi sensor.



Gambar 3.6. Diagram blok MUX

Gambar 3.7. Diagram blok DFF

Gambar 3.8. Diagram blok *Dividing Frequency circuit*

Pada diagram blok DFF, pin input inp dan clk dapat digunakan secara bergantian, karena ouput (q) akan mengeluarkan selisih frekuensi dari kedua input. Sesuai dengan persamaan berikut

$$fd = |fr - fx|$$

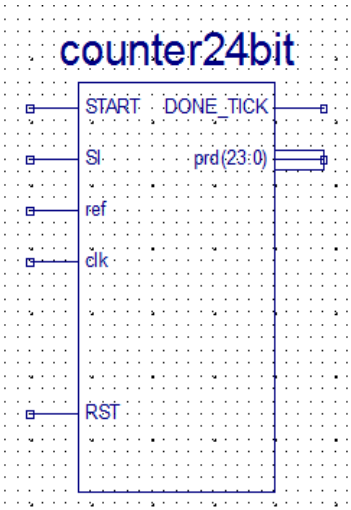
Dividing Frequency circuit (sebagai pembagi frekuensi), digunakan untuk memperpanjang pulsa frekuensi, yang akan digunakan untuk periode pada pencacahan frekuensi.

Pin input fx digunakan untuk masukkan frekuensi yang akan dibagi serta dikeluarkan pada pin output fx_perN. Pin clk sebagai clock dan pin RST sebagai reset.

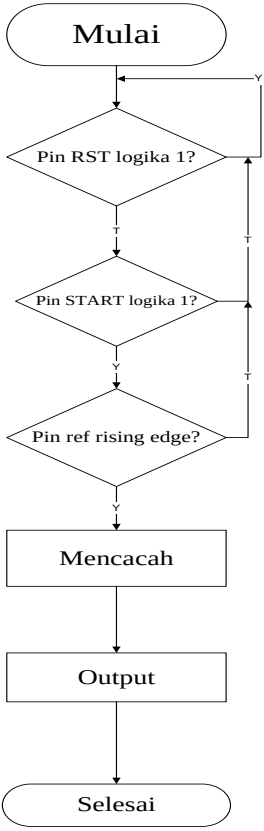
counter (untuk mencacah frekuensi dengan metode *reciprocal frequency counter*) akan melakukan pencacahan frekuensi referensi setiap periode frekuensi sensor yang telah melewati proses pengurangan dengan frekuensi referensi dan proses pembagian untuk memperpanjang pulsanya. Sesuai dengan persamaan berikut

$$C = \frac{fr \cdot N}{fd}$$

Proses pencacahan tidak akan dimulai bila pin input START tidak mendapatkan logika 1. Pencacahan dilakukan dengan mencacah banyaknya logika 1 yang masuk pada pin input ref setiap periode masukan pin input SI. Pin clk sebagai clock dan pin RST sebagai reset. Hasil dari pencacahan akan di keluarkan pada pin output prd(23:0) (24 bit). Ketika proses pencacahan selesai, pin output DONE_TICK akan memberikan logika 1.



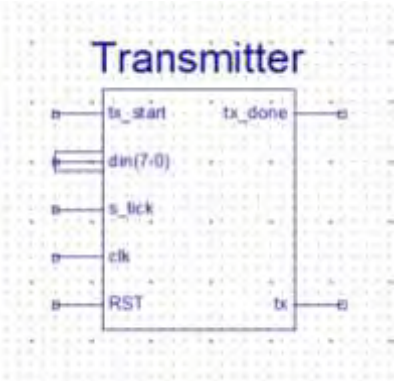
Gambar 3.9. Diagram blok *counter* 24bit



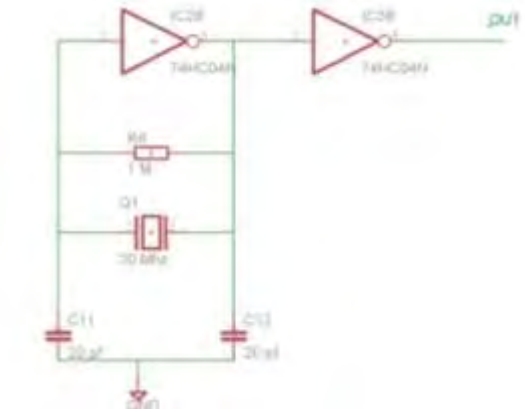
Gambar 3.10. *Flowchart* rangkaian counter

Untuk pengiriman data ke PC, menggunakan komunikasi serial. Komunikasi ini harus sesuai dengan *standard interface* komunikasi dari PC. Pada tugas akhir ini menggunakan *interface* RSS-232.

Proses pengiriman data dari FPGA ke PC akan memberikan logika 1 ketika *idle*, kemudia mengirim logika 0 (sebagai sinyal start) sebelum pengiriman datanya. Kemudian data (8bit) dikirim dan diakhiri dengan logika 1 (sebagai sinyal stop) dan kembali lagi ke *idle*.



Gambar 3.11. Diagram blok Transmitter



Gambar 3.12. Rangkaian osilator

Pada FPGA dibuat diagram blok transmitter untuk pengiriman data secara serial ke PC. Blok ini tidak akan mengirim data, kecuali pin input tx_start diberikan logika 1. Pin input din(7:0) (8bit) akan dikirim secara serial pada pin output tx dengan kecepatan aliran data (*baud rate*) di FPGA pada pin input s_tick yang telah disamakan dengan PC. Selesai pengiriman data, akan memberikan logika 1 pada pin output tx_done.

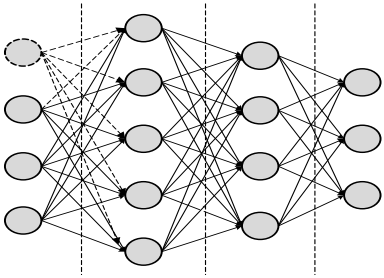
Rangkaian osilator yang digunakan adalah osilator pierce, osilator ini menggunakan kristal sebagai rangkaian tangkinya, seperti ditunjukkan pada gambar 3.12.

Pada osilator ini Kristal merespon sebagai rangkaian resonansi parallel. Keluaran pengoperasian osilator didasarkan pada balikan yang dipasang antara C1 dan C2. Keluaran dari inverter akan mengalami pembalikan fase. Nilai C1 dan C2 menentukan besarnya tegangan balikan. Sekitar 10-50% dari keluaran dikirim kembali sebagai balikan untuk memberikan energi ke Kristal. Jika Kristal mendapatkan energi yang tepat, frekuensi resonansi yang dihasilkan akan sangat tajam. Kristal akan bergetar pada selang frekuensi yag sangat sempit. Output dari osilator ini sangat stabil. Namun output dari osilator pierce ini sangat kecil dan Kristal dapat mengalami kerusakan dengan strain mekanik yang terus menerus.

Perangkat lunak pada PC menggunakan Delphi 7, yang terdiri dari pemrograman penerimaan data serial dan proses *learning* pengidentifikasian gas. Pengidentifikasian gas dapat dilakukan apabila sistem telah mendapatkan topologi *neural network* hasil *training* yang digunakan untuk memproses data yang didapat dari sampel gas.

Komunikasi serial pada delphi digunakan sebagai pengirim sinyal pada FPGA untuk memulai dan menghentikan proses pencacahan serta untuk penerima data yang dikirim oleh FPGA. Pada komunikasi ini, diatur *baud rate* dan jumlah bit data yang dikirim atau diterima disamakan pada FPGA.

Topologi *neural network* yang digunakan terdiri dari empat layer, yaitu layer input, dua hidden layer, dan satu layer output. Sesuai dengan gambar 3.13.



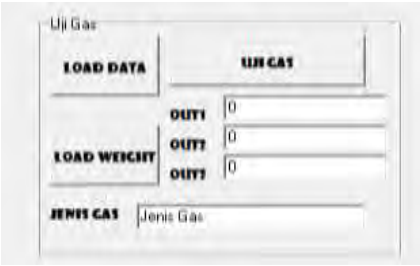
Gambar 3.13. Topologi *neural network*

Tampilan pembacaan frekuensi tiap sensor ditunjukkan pada gambar 3.13. Hasil pembacaan frekuensi ini akan disimpan dan

dilakukan *training* untuk mendapatkan *weight* dan *threshold* dari tiap layer.Tampilan data *training* ditunjukkan pada gambar 3.15.

Gambar 3.14. Tampilan pembacaan frekuensi tiap sensor

Gambar 3.15. Tampilan data *training*



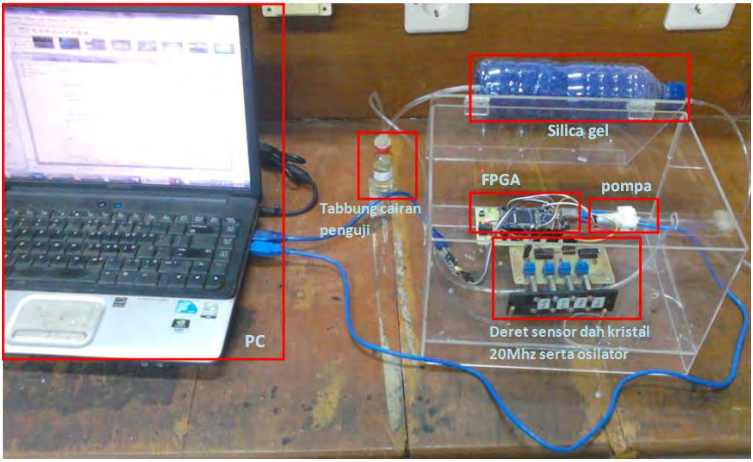
Gambar 3.16. Tampilan data *learning*

Weight dan *threshold* diperoleh ketika nilai *error* terkecil didapatkan. Proses selanjutnya adalah proses *learning* dan pengidentifikasian gas. Tampilan data *learning* ditunjukkan pada gambar 3.16. Proses ini menggunakan gas uji yang akan

diidentifikasi dengan *weight* dan *threshold* yang di dapat dari proses *training*.

BAB IV
PENGUJIAN DAN PEMBAHASAN SISTEM

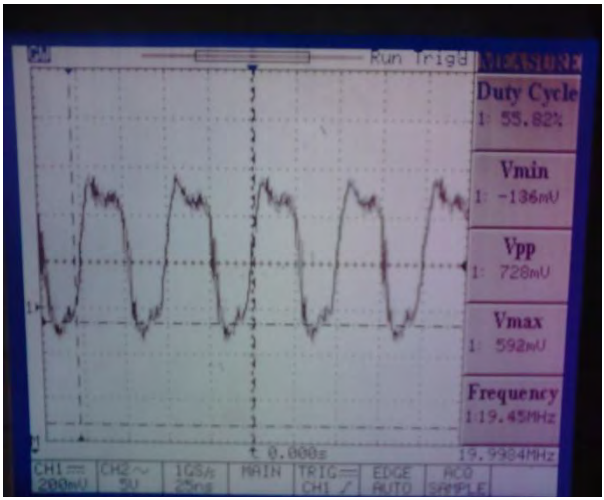
Bab ini membahas tentang perancangan sistem yang sudah dirancang, dimana hasil rancangan sistem tersebut akan diuji dan dibahas. Cara pengujian dan pembahasan pada bab ini adalah dengan membahas tiap blok dari perancangan sistem secara keseluruhan dengan disertai tabel atau gambar yang mendukung pengujian dan pembahasan sistem. Realisasi blok diagram sistem ditunjukkan pada gambar 4.1.



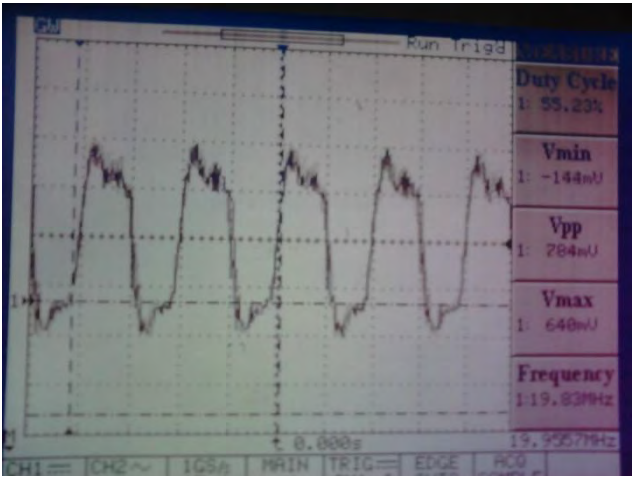
Gambar 4.1. Realisasi blok diagram sistem

4.1. Pengujian Rangkaian Osilator

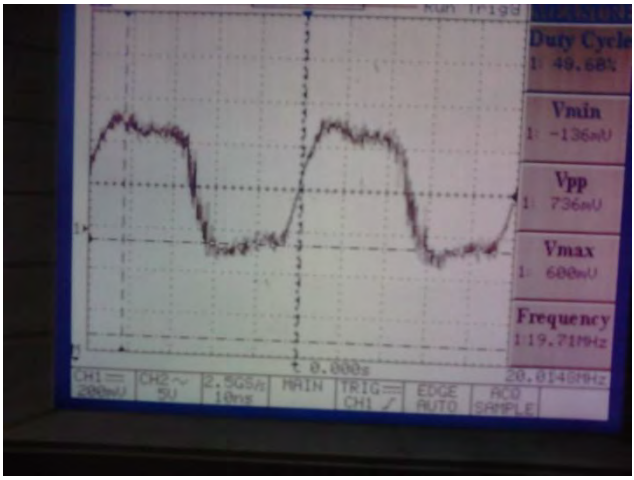
Pengujian ini dilakukan dengan semua rangkaian osilator sensor dan referensi, yang bertujuan untuk memastikan rangkaian berfungsi dengan semestinya. Pengujian dilakukan dengan menghubungkan output dari rangkaian dengan osiloskop. Dari hasil pengujian rangkaian osilator referensi, menunjukkan sinyal berbentuk *square* tidak sempurna, ditunjukkan pada gambar 4.2



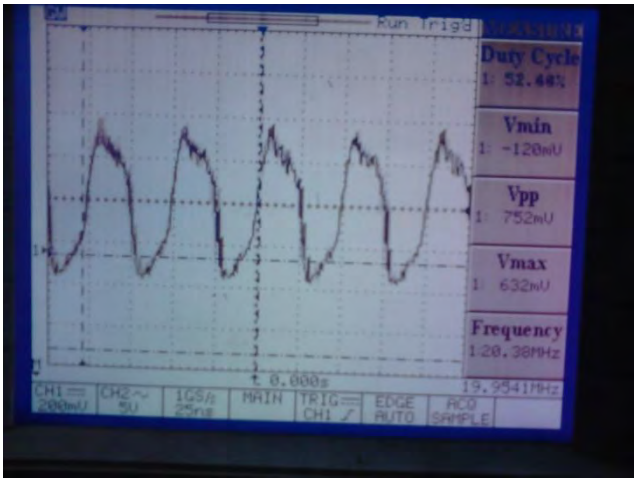
Gambar 4.2. Output osilator 20Mhz



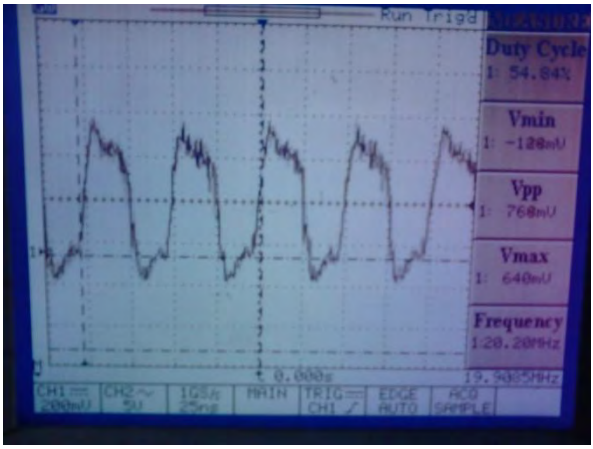
Gambar 4.3. Output osilator sensor PEG-1540



Gambar 4.4 Output osilator sensor OV-275



Gambar 4.5. Output osilator sensor Cellulose



Gambar 4.6. Output osilator sensor Ap-L

Sinyal yang tampak pada osiloskop berbentuk *square* tidak sempurna, hal ini disebabkan karena *delay* dari komponen yang digunakan. Dari gambar 4.3, gambar 4.4, gambar 4.5, gambar 4.6, terlihat bahwa output rangkian osilator tiap sensor sekitar 20Mhz. *Error* rata-rata yang didapatkan sekitar 0,001981%. Dari hasil pengujian dapat diketahui bahwa rangkaian osilator dapat berfungsi dengan baik.

4.2. Pengujian Rangkaian *Frequency difference* Pada FPGA

Pengujian ini dilakukan dengan cara memberikan dua sinyal frekuensi menggunakan *function generator* yang dihubungkan pada pin input FPGA yang telah diisi program rangkaian *frequency difference* dan pin output pada osiloskop. Rangkaian *frequency difference* ini menggunakan prinsip kerja rangkaian logika DFF, yang mana akan memberi keluaran seperti pin D ketika pin clk berstatus *rising edge*. Hasil yang didapat dapat dilihat pada tabel 4.1.

Tabel 4.1. Pengujian rangkaian *frequency difference*

D(Mhz)	CLK(Mhz)	Out(Mhz)	error
20	19	0.99997	0.003%
20	18	1.996	0.050%
20	11	8.999	0.011%

D(Mhz)	CLK(Mhz)	Out(Mhz)	error
20	10	9.5	5.000%
20	9	8.999	18.191%
19	20	0.99997	0.003%
15	20	4.999	0.020%
11	20	8.999	0.011%
10	20	9.5	5.000%
9	20	8.999	18.191%

Keseluruhan tabel dapat dilihat pada lampiran tabel 1a. Dari hasil pengujian, dapat disimpulkan rangkaian *frequency difference* yang diimplementasikan pada FPGA berfungsi dengan baik untuk beda frekuensi antara 1 Mhz sampai 9 Mhz. Pada pengujiannya rangkaian ini memiliki *error* rata-rata 14,53 %.

4.3. Pengujian Rangkaian Pembagi Frekuensi Pada FPGA

Pengujian ini dilakukan sama seperti pengujian rangkaian *frequency difference*, yaitu dengan menghubungkan pin FPGA yang telah diisi program rangkaian pembagi frekuensi pada osiloskop. Hasil yang didapat dapat dilihat pada tabel 4.2.

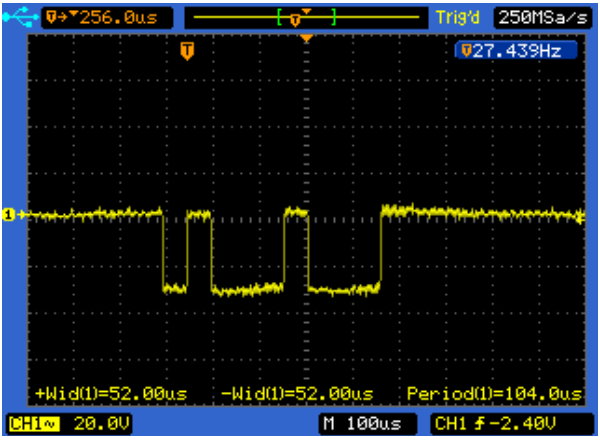
Tabel 4.2. Pengujian rangkaian pembagi frekuensi

input (kHz)	dividing by	output (Hz)	output seharusnya (Hz)	error(%)
9.9538	32	310	311.05625	0.34%
9.9538	64	153	155.528125	1.63%
9.9538	128	78	77.7640625	0.30%
9.9538	256	38	38.88203125	2.27%
9.9538	512	20	19.44101563	2.88%
9.9538	1024	10	9.720507813	2.88%
9.9538	2048	5	4.860253906	2.88%

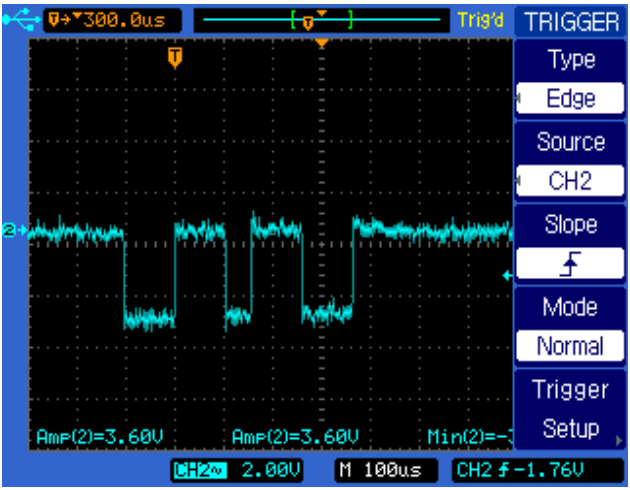
Dari hasil pengujian dapat disimpulkan rangkaian pembagi frekuensi yang diimplementasikan pada FPGA berfungsi dengan baik. Pada pengujiannya rangkaian ini memiliki *error* rata-rata sebesar 1,88%. Karena output dari rangkaian ini akan digunakan sebagai periode lamanya pencacahan, maka pemilihan angka pembagi sangat berpengaruh pada hasil pencacahan, semakin besar pembaginya maka hasil pencacahan akan semakin besar sesuai dengan metode *reciprocal frequency counter*. Semakin besar nilai pembaginya, maka resolusi pencacahan akan semakin tinggi.

4.4. Pengujian Komunikasi Serial

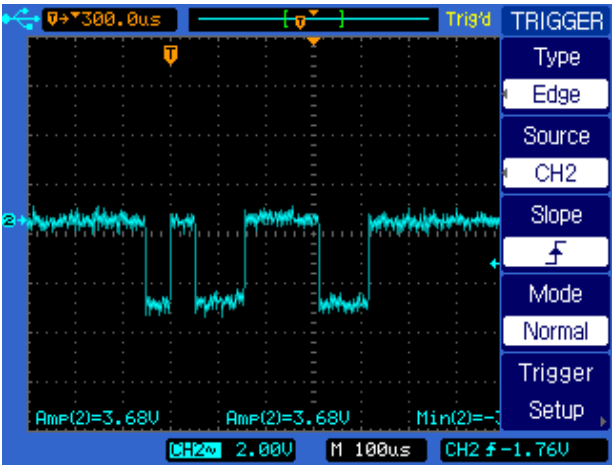
Pengujian ini dilakukan dengan cara menghubungkan pin output FPGA yang telah diisi program rangkaian transmitter pada osiloskop. Dengan diberi masukan menggunakan program Hterm 0.7.1 yang dikirim secara terus menerus. Sebelum melakukan proses pengiriman data, output dari blok transmitter ini akan berlogika 1 (*idle*), ketika ada data yang akan dikirim, akan memberikan logika 0 (start), kemudian barulah 8bit data dikirim dengan LSB (*Least Significant Bit*) terlebih dahulu dan diikuti dengan bit selanjutnya hingga MSB (*Most Significant Bit*). Setelah bit data, akan kembali *idle*. Hasil pengujian dapat dilihat pada gambar 4.7, gambar 4.8, gambar 4.9.



Gambar 4.7. Pengiriman data 0001 0001



Gambar 4.8. Pengiriman data 0011 0110



Gambar 4.9. Pengiriman data 0011 1001

Dari hasil pengujian, dapat disimpulkan blok trasmitter yang diimplementasikan pada FPGA dapat berfungsi dengan baik.

4.5. Pengujian Rangkaian Pencacah Frekuensi

Pada pengujian ini dilakukan pencacahan metode *reciprocal*. Dengan menggunakan dua sinyal frekuensi yang berbeda, satu sinyal digunakan sebagai frekuensi dan sinyal lain sebagai periode pencacahan. Hasil pencacahan yang didapat dapat dilihat pada tabel 4.3.

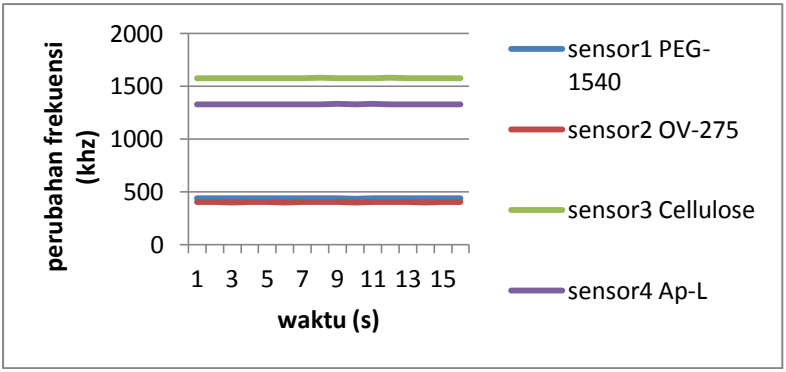
Tabel 4.3. Frekuensi tiap sensor yang terbaca

Sinyal 1 (sebagai frekuensi(Mhz))	Sinyal 2 (sebagai periode pencacahan (µs))	Hasil pencacahan	error (%)
20	10^{-6}	20	0.00%
20	$0.5*10^{-6}$	10	0.00%
20	$0.33*10^{-6}$	6	0.00%
20	$0.25*10^{-6}$	5	0.00%
20	$0.2*10^{-6}$	4	0.00%
20	$0.166*10^{-6}$	3	0.00%
20	$0.143*10^{-6}$	2	0.00%
20	$0.125*10^{-6}$	2	0.00%
20	$0.11*10^{-6}$	2	0.00%
20	10^{-5}	1	50.00%

Adanya *error* pada periode 10^{-5} µs, dikarenakan pergeseran frekuensi yang tidak tepat pada periode tersebut. Pada sinyal 2 periode 10^{-5} µs memiliki frekuensi 10 Mhz yang berarti setengah dari sinyal 1. Rangkaian pencacah frekuensi yang diimplementasikan pada FPGA ini dapat berfungsi dengan baik bila sinyal frekuensi yang digunakan sebagai periode pencacahan tidak kurang dari setengah frekuensi yang di cacah. Pada pengujiannya rangkaian ini memiliki *error* rata-rata sebesar 5%.

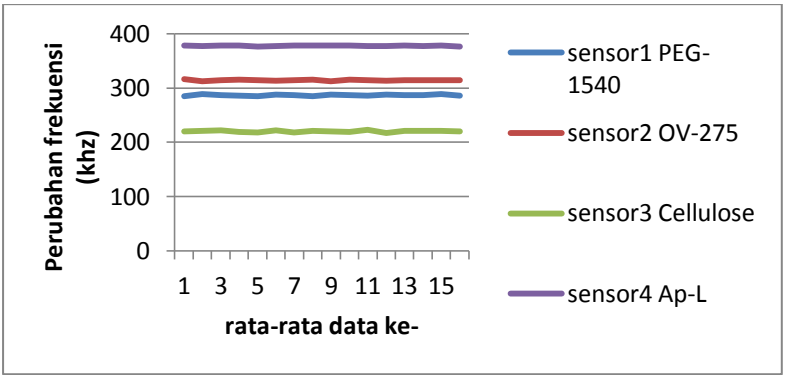
4.6. Pengujian Sensor

Pengujian ini dilakukan dengan cara memberikan beberapa sampel gas pada deret sensor QCM. Gas uji yang digunakan adalah amoniak, minyak kayu putih, dan thinner. Setelah mendapatkan perubahan frekuensi tiap sensor, dicari rata-ratanya tiap 100 data perubahan frekuensi yang diperoleh.



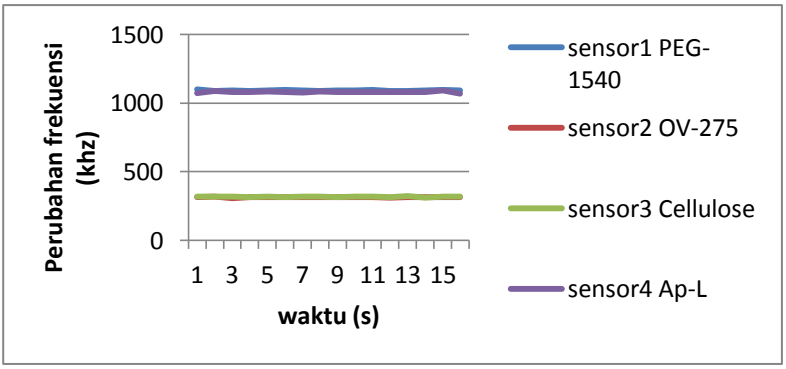
Gambar 4.10. Grafik perubahan frekuensi terhadap amoniak

Dari gambar 4.10, dapat disimpulkan bahwa respon sensor3 cellulose dan sensor4 Ap-L yang paling sensitif terhadap jenis gas amoniak.



Gambar 4.11. Grafik perubahan frekuensi terhadap thinner

Dari gambar 4.11, dapat disimpulkan bahwa respon tiap sensor tidak terlalu besar terhadap jenis gas thinner.



Gambar 4.12. Grafik perubahan frekuensi terhadap minyak kayu putih

Dari gambar 4.12, dapat disimpulkan bahwa respon sensor1 PEG-1540 dan sensor4 Ap-L yang paling sensitif terhadap jenis gas minyak kayu putih.

4.7. Pengujian Neural Network

Pada pengujian *neural network* digunakan *software* delphi untuk mendapatkan nilai *weight* dan *threshold*. Pengujian ini dilakukan dengan memberikan deret sensor beberapa masukan sampel gas. Gas yang digunakan dalam pengujian sensor adalah minyak kayu putih, amoniak, dan thinner. Pada saat proses *training*, *error* yang diinginkan adalah dibawah 0,0001. Apabila nilai *error* telah mencapai yang diinginkan, nilai *weight* dan *threshold* yang akan digunakan untuk pengenalan gas akan didapatkan.

Perolehan *error* ketika iterasi ke-1268. *Error* yang didapatkan pada proses *learning* dari tiap gas, dapat dilihat pada tabel 4.4.

Tabel 4.4. Nilai *error* pada proses *training neural network*

	amoniak	Thinner	m.kayu putih
error	$3.995182*10^{-5}$	$9.997505*10^{-5}$	$7.09260*10^{-5}$

Pengujian dilakukan sebanyak 10 kali tiap gas uji, didapatkan hasil sebagai berikut:

Tabel 4.5. Pengujian *neural network* pada gas uji

uji ke-	gas yang di uji	Hasil pembacaan
1	Amoniak	Amoniak
2	Amoniak	Amoniak
3	Amoniak	Amoniak
4	Amoniak	Amoniak
5	Amoniak	Amoniak
6	Amoniak	Amoniak
7	Amoniak	Amoniak
8	Amoniak	Amoniak
9	Amoniak	Amoniak
10	Amoniak	Amoniak
11	Thinner	Thinner
12	Thinner	Amoniak
13	Thinner	M. Kayu Putih
14	Thinner	Thinner
15	Thinner	M. Kayu Putih
16	Thinner	Amoniak
17	Thinner	Amoniak
18	Thinner	Thinner
19	Thinner	Thinner
20	Thinner	Thinner
21	M. Kayu Putih	M. Kayu Putih
22	M. Kayu Putih	M. Kayu Putih
23	M. Kayu Putih	M. Kayu Putih
24	M. Kayu Putih	M. Kayu Putih
25	M. Kayu Putih	Thinner
26	M. Kayu Putih	M. Kayu Putih
27	M. Kayu Putih	M. Kayu Putih

uji ke-	gas yang di uji	Hasil pembacaan
28	M. Kayu Putih	M. Kayu Putih
29	M. Kayu Putih	M. Kayu Putih
30	M. Kayu Putih	M. Kayu Putih

Pada Tabel 4.5 terlihat kesalahan pengidentifikasian gas. Dari 30 pengujian, terdapat 6 kesalahan, Proses identifikasi gas uji memiliki prosentase keberhasilan 80%.

4.8. Hasil Pengujian data

Dari pengujian yang dilakukan dapat disimpulkan rangkaian osilator tiap sensor dan referensi berfungsi dengan baik, walaupun terdapat *error* 0,001981%.

Skema rangkaian yang diimplementasikan pada FPGA dapat berfungsi dengan baik, mulai dari rangkaian *frequency difference*, pembagi frekuensi, rangkaian counter, dan komunikasi serial ke PC. Rangkaian *frequency difference* berfungsi dengan baik, bila beda frekuensi antara 1 Mhz sampai 9 Mhz. Oleh karena itu digunakan frekuensi referensi dan frekuensi sensor yang besarnya antara 19 Mhz sampai 20 Mhz. Pada pengujiannya rangkaian *frequency difference* memiliki *error* rata-rata 14,53%. Rangkaian pembagi frekuensi berfungsi dengan baik, rangkaian ini akan digunakan sebagai periode lamanya pencacahan, semakin besar nilai pembaginya maka frekuensi output dari rangkaian ini akan semakin kecil, sehingga periodenya semakin besar Pada pengujiannya rangkaian pembagi frekuensi memiliki *error* rata-rata 1,88 %, Rangkaian pencacah frekuensi berfungsi dengan baik bila beda frekuensi sebagai periode tidak lebih dari dua kali frekuensi yang dicacah. Pada pengujiannya rangkaian pencacah frekuensi memiliki nilai *error* sebesar 5%. Dari pengujian Komunikasi serial menggunakan rangkaian transmitter berfungsi dengan baik sesuai dengan teori komunikasi secara serial.

Dalam pengenalan gas uji, terdapat 6 kesalahan indentifikasi gas dari 30 pengujian. Tingkat keberhasilan dari keseluruhan sistem mencapai 80%. Kesalahan identifikasi gas diperkirakan karena sensor kurang sensitif dalam menanggapi gas uji.

BAB V
PENUTUP

5.1. Kesimpulan

Pada penelitian ini telah dirancang pencacahan frekuensi dengan metode reciprocal untuk sensor gas resonator kuarsa yang diimplementasikan pada *Field Programmable Gate Array*. Pada rangkaian *frequency difference* akan berfungsi dengan baik bila beda frekuensi antara 1 Mhz sampai 9 Mhz. Pemilihan nilai pembagi pada rangkaian pembagi frekuensi sangat mempengaruhi hasil dan lama proses pencacahan, semakin besar nilai pembaginya maka resolusi pencacahan akan semakin tinggi. Pada rangkaian *counter*, sinyal frekuensi yang digunakan sebagai periode pada proses pencacahan tidak kurang dari setengah frekuensi yang diacacah. Proses identifikasi gas uji menggunakan *neural network*, mampu mengidentifikasi gas uji dengan tingkat keberhasilan pada keseluruhan sistem mencapai 80%.

5.2. Saran

Pada pembuatan dan pengujian tugas akhir ini terdapat beberapa kekurangan dan disarankan untuk menggunakan sensor yang lebih sensitif dalam menanggapi gas dan menambah jumlah deret sensor, agar gas dapat diidentifikasi lebih baik. Serta dapat merancang *frequency difference* dengan besar frekuensi yang tidak ada batasan frekuensi.

LAMPIRAN

TABEL 1a

D(Mhz)	CLK(Mhz)	Out(Mhz)	error
20	19	0.99997	0.00%
20	18	1.996	0.05%
20	17	2.999	0.03%
20	16	3.99	0.03%
20	15	4.999	0.02%
20	14	5.999	0.02%
20	13	6.999	0.01%
20	12	7.999	0.01%
20	11	8.999	0.01%
20	10	9.5	5.00%
20	9	8.999	18.19%
20	8	7.999	33.34%
20	7	6.999	46.16%
20	6	5.999	57.15%
20	5	4.999	66.67%
20	4	3.999	75.01%
20	21	0.998	0.20%
20	22	1.995	0.25%
20	23	2.988	0.40%
20	24	3.999	0.02%
20	25	4.989	0.22%
20	26	5.998	0.03%
20	27	6.988	0.17%
20	28	7.989	0.14%
20	29	8.889	1.23%
20	30	9.6	4.00%
19	20	0.99997	0.00%
18	20	1.996	0.05%
17	20	2.999	0.03%
16	20	3.99	0.03%
15	20	4.999	0.02%
14	20	5.999	0.02%

13	20	6.999	0.01%
12	20	7.999	0.01%
11	20	8.999	0.01%
10	20	9.5	5.00%
9	20	8.999	18.19%
8	20	7.999	33.34%
7	20	6.999	46.16%
6	20	5.999	57.15%
5	20	4.999	66.67%

LISTING PROGRAM VHDL

FREQUENCY DIFFERENCE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity DFF is
 Port (
          d          : in  STD_LOGIC;
        clk          : in  STD_LOGIC;
      reset          : in  STD_LOGIC;
        q                       : out  STD_LOGIC
                    );
end DFF;
architecture Behavioral of DFF is
begin
          process (clk,reset)
          begin
                    if (reset = '1') then
                              q <= '0';
                    elsif (rising_edge (clk)) then
                              q <= d ;
                    end if;
          end process;
end Behavioral;
```

PEMBAGI FREKUENSI

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
entity dividing_Freq is
generic
(
        div     : integer := 256            -- N = div
);
port
(
        RST             : in std_logic;
        fx              : in std_logic;
        clk             : in    std_logic;
        fx_perN : out std_logic
);
end dividing_Freq;

architecture Behavioral of dividing_Freq is
type state is (IDLE, DIVIDE);
constant N : integer:= (div/2);
signal p_state, n_state : state;
signal clktemp_next, clktemp_reg   : std_logic:='0';
signal edge                        : std_logic;
signal delay_reg                   : std_logic;
signal b_reg,b_next                : unsigned (31 downto 0);

begin
fx_perN <= clktemp_reg;
process(RST, clk)
begin
                if RST = '1' then
                        p_state <= IDLE;
                        b_reg           <= (others => '0');
                        delay_reg <= '0';
                        clktemp_reg <= '0';
                elsif clk'event and clk = '1' then
                        p_state <= n_state;
                        delay_reg <= fx;
                        b_reg <= b_next;
                        clktemp_reg <= clktemp_next;
                end if;
end process;
```

```
edge <= (not delay_reg) and fx;
process(edge, p_state, b_reg, clktemp_reg)
begin
        n_state <= p_state;
        b_next <= b_reg;
        clktemp_next <= clktemp_reg;
        case p_state is
        when IDLE =>
                        if (edge ='1') then
                                n_state <= DIVIDE;
                        end if;

        when DIVIDE =>
                        if (edge = '1') then
                                if (b_reg = N-1) then
                          clktemp_next <= not clktemp_reg;
                          b_next  <= (others=> '0');
                          else
                          b_next <= b_reg +1;
                                end if;
                         end if;

        end case;
end process;
end Behavioral;
```

COUNTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity counter24bit is
generic
(
        data_length : integer := 8
);
port
(
        RST                                     : in  std_logic;  --
reset
```

```
        ref                                     : in std_logic; --
clock external refrensi
        clk                                     : in std_logic; --
clock FPGA
        DONE_TICK                       : out std_logic;
        SI, START                       : in std_logic; --signal for
"input" and "ready" for counting
        prd                                     :               out
std_logic_vector(23 downto 0)--(data_length-1 downto 0) --periods +2
header
);
end counter24bit;

architecture Behavioral of counter24bit is

type state is (IDLE, WAITING, COUNT, DONE);
signal p_state, n_state : state;
signal b_reg, b_next            : unsigned(23 downto 0);
signal delay_reg                : std_logic;
signal delay_reg2               : std_logic;
signal edge, edge2              : std_logic;


begin
process(RST,clk)
begin
        if RST = '1' then
                p_state         <= IDLE;
                b_reg           <= (others=>'0');
                delay_reg       <= '0';
                delay_reg2      <= '0';
        elsif clk'event and clk='1' then
                p_state         <= n_state;
                b_reg           <= b_next;
                delay_reg       <= SI;
                delay_reg2      <= ref;
        end if;
end process;
```

```
edge <= (not delay_reg) and SI;
edge2 <= (not delay_reg2) and ref;

---==  FSM
process(START, edge, edge2, p_state, b_reg) --, hd_next, hd_reg)
begin
--                  RDY <= '0';
          DONE_TICK <= '0';
          n_state <= p_state;
          b_next <= b_reg;

case p_state is
          when IDLE =>
--                            RDY <= '1';
                    if START = '1' then
                              n_state <= WAITING;
                    end if;
          when WAITING =>
                    if (edge = '1') then
--                                  RDY <= '1';
                              n_state <= COUNT;
                              b_next <= (others =>'0');
--                                  b_next <= 0;
                    end if;
          when COUNT =>
                    if (edge ='1') then
                              n_state <= DONE;
                              done_tick <= '1';


                    elsif (edge2 ='1') then
                                        b_next <= b_reg + 1;
                              end if;
          when DONE =>
                    n_state    <= IDLE;

          end case;
end process;

prd <= std_logic_vector(b_reg);
```

```
end Behavioral;
```

KOMUNIKASI SERIAL (TRANSMITTER DAN RECEIVER)

TRANSMITTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Transmitter is
generic
	(
		DBIT		: integer := 8;
		SB_TICK		: integer := 16
	);
port
	(
		clk, RST: in std_logic;
		tx_start  : in std_logic;
		s_tick    : in std_logic;
		din		: in std_logic_vector(7 downto 0);
		tx_done : out std_logic;
		tx			: out std_logic
	);
end Transmitter;

architecture Behavioral of Transmitter is
type state is (IDLE, START, DATA, STOP);
signal state_reg, state_next		: state;
signal s_reg, s_next				:	unsigned(3
downto 0);
signal n_reg, n_next				:	unsigned(2
downto 0);
signal b_reg, b_next			:	std_logic_vector(7
downto 0);
signal tx_reg, tx_next			: std_logic;

begin
```

```
--FSM state & data register
process(clk,RST)
begin
	if RST='1' then
		state_reg <= IDLE;
		s_reg <= (others =>'0');
		n_reg <= (others =>'0');
		b_reg <= (others =>'0');
		tx_reg <= '1';
	elsif (clk'event and clk='1') then
		state_reg <= state_next;
		s_reg <= s_next;
		n_reg <= n_next;
		b_reg <= b_next;
		tx_reg <= tx_next;
	end if;
end process;

process(state_reg, s_reg, n_reg, b_reg,
				s_tick, tx_reg, tx_start, din)
begin
	state_next <= state_reg;
	s_next  <= s_reg;
	b_next  <= b_reg;
	n_next  <= n_reg;
	tx_next <= tx_reg;
	tx_done <= '0';

	case state_reg is
		when IDLE =>
			tx_next <= '1';
			if tx_start = '1' then
				state_next <= START;
				s_next <= (others => '0');
				b_next <= din;
			end if;
		when START =>
			tx_next <= '0';
			if (s_tick = '1') then
				if s_reg = 15 then
```

```
                                        state_next <= DATA;
                                        s_next <= (others => '0');
                                        n_next <= (others => '0');
                                else
                                        s_next <= s_reg +1;
                                end if;
                        end if;
                when DATA =>
                        tx_next <= b_reg(0);
                        if (s_tick = '1') then
                                if s_reg = 15 then
                                        s_next <= (others =>'0');
                                b_next <= '0' & b_reg(7 downto 1);
                                        if n_reg = (DBIT - 1) then
                                                state_next      <=
STOP;
                                        else
                                        n_next <= n_reg + 1;
                                        end if;
                                else
                                        s_next <= s_reg + 1;
                                end if;
                        end if;
                when STOP =>
                        tx_next <= '1';
                        if (s_tick = '1') then
                                if s_reg = (SB_TICK-1) then
                                        state_next <= IDLE;
                                        tx_done <= '1';
                                else
                                        s_next <= s_reg + 1;
                                end if;
                        end if;
                end case;
end process;

tx <= tx_reg;
end Behavioral;
```

RECEIVER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Receiver is
generic
(
        DBIT            : integer := 8;
        SB_TICK         : integer := 16
);
port
(
        RST             : in std_logic;
        clk             : in std_logic;
        Rx                      : in std_logic;
        s_tick    : in std_logic;
        done    : out std_logic;
        Dout            : out std_logic_vector(7 downto 0)
);
end Receiver;

architecture Behavioral of Receiver is
type state_type is (idle, start, data, stop);
signal state_reg, state_next : state_type;
signal s_reg, s_next : unsigned(3 downto 0);
signal n_reg, n_next : unsigned(2 downto 0);
signal b_reg, b_next : std_logic_vector(7 downto 0);
begin
process(RST, clk)
begin
        if RST = '1' then
                state_reg <= idle;
                s_reg <= (others=>'0');
                b_reg <= (others=>'0');
                n_reg <= (others=>'0');
        elsif clk'event and clk='1' then
                state_reg <= state_next;
                s_reg <= s_next;
                b_reg <= b_next;
                n_reg <= n_next;
```

```
        end if;
end process;

process(s_tick, Rx, s_reg, b_reg, n_reg, state_reg)
begin
        state_next <= state_reg;
        s_next <= s_reg;
        b_next <= b_reg;
        n_next <= n_reg;
        done <= '0';

        case state_reg is
                when idle =>
                        if Rx ='0' then
                                state_next <= start;
                                s_next <= (others=>'0');
                        end if;
                when start =>
                        if s_tick = '1' then
                                if s_reg = 7 then
                                        state_next <= data;
                                        s_next <= (others=>'0');
                                        n_next <= (others=>'0');
                                else
                                        s_next <= s_reg + 1;
                                end if;
                        end if;
                when data =>
                        if s_tick='1' then
                                if s_reg = 15 then
                        s_next <= (others=>'0');
                        b_next <= Rx & b_reg(7 downto 1);
                        if n_reg = DBIT-1 then
                        state_next <= stop;
                        else
                        n_next <= n_reg + 1;
                        end if;
                        else
                        s_next <= s_reg + 1;
                        end if;
```

```
                    end if;
                when stop =>
                    if s_tick ='1' then
                        if s_reg = SB_TICK-1 then
                            state_next <= idle;
                            done <= '1';
                        else
                            s_next <= s_reg + 1;
                        end if;
                    end if;
        end case;
Dout <= b_reg;
end process;

end Behavioral;
```

LISTING PROGRAM DELPHI

```
//Penerimaan data comport
data1:= integer(DataTerima[1]);
data2:= integer(DataTerima[2]);
data3:= integer(DataTerima[3]);

//Proses mendapatkan frekuensi
sensor[isen]:= (data3 *65536) + (data2 *256) + data1;

   isen := isen +1;

   if isen = 5 then
   begin
    if (sensor[1] <> 0) then freksen1[iterasi] := (1-
(Npembagi/sensor[1]))*20
    else freksen1[iterasi] := freksen1[iterasi -1];

    if (sensor[2] <> 0) then freksen2[iterasi] := (1-
(Npembagi/sensor[2]))*20
    else freksen2[iterasi] := freksen2[iterasi -1];
```

```
    if (sensor[3] <> 0) then freksen3[iterasi] := (1-
(Npembagi/sensor[3]))*20
    else freksen3[iterasi] := freksen3[iterasi -1];

    if (sensor[4] <> 0) then freksen4[iterasi] := (1-
(Npembagi/sensor[4]))*20
    else freksen4[iterasi] := freksen4[iterasi -1];

//neural network
procedure TForm1.SpeedButton12Click(Sender: TObject);
var
  a,b,c,d : integer;

begin

  for b:= 1 to layer1 do
  begin
    buff_nn := 0;
    for a:= 1 to layer0 do
    begin
      buff_nn := buff_nn + (in_2[a]*w_1[a,b]);
    end;
    buff_nn       := buff_nn - trs1[b];
    f_v1[b]    := 1/(1+(exp(-alpha*buff_nn)));
    g_v1[b]    := alpha*f_v1[b]*(1-f_v1[b]);
  end;

  for c:= 1 to layer2 do
  begin
    buff_nn:= 0;
    for b:= 1 to layer1 do
    begin
      buff_nn := buff_nn + (f_v1[b]*w_2[b,c]);
    end;
    buff_nn := buff_nn - trs2[c];
    f_v2[c] := 1/(1+(exp(-1*alpha*buff_nn)));
    g_v2[c] := alpha*f_v2[c]*(1-f_v2[c]);
  end;
```

```
 for d:= 1 to layer3 do
 begin
  buff_nn:= 0;
  for c:= 1 to layer2 do
  begin
    buff_nn := buff_nn + (f_v2[c]*w_3[c,d]);
  end;
  buff_nn := buff_nn - trs3[d];
  f_v3[d]        := 1/(1+(exp(-1*alpha*buff_nn)));
  g_v3[d]        := alpha*f_v3[d]*(1-f_v3[d]);
 end;
 //f_v3[3]:=9 div 4;
 maskedit4.Text:=floattostr(f_v3[1]);
 maskedit5.Text:=floattostr(f_v3[2]);
 maskedit6.Text:=floattostr(f_v3[3]);

 //if (f_v3[1]>0.23) and (f_v3[2]<0.22) and (f_v3[3]<0.28)        //85 15
15
 if (f_v3[1]>f_v3[2]) and (f_v3[1]>f_v3[3]) //and (f_v3[1]>0.23) and
(f_v3[2]<0.25) and (f_v3[3]<0.28)
 then Maskedit3.Text := 'AMONIA'

 //else if (f_v3[1]<0.42) and (f_v3[2]>0.20) and (f_v3[3]<0.75)     //15
20 75
 else if (f_v3[2]>f_v3[1]) and (f_v3[2]>f_v3[3])// and (f_v3[1]<0.72)
and (f_v3[2]>0.20) and (f_v3[3]<0.79)
 then Maskedit3.Text := 'M. KAYU PUTIH'

 //else if (f_v3[1]<0.25) and (f_v3[2]<0.350) and (f_v3[3]>0.4)      //15
15 70
 else if (f_v3[3]>f_v3[1]) and (f_v3[3]>f_v3[2])// and (f_v3[1]<0.25)
and (f_v3[2]<0.35) and (f_v3[3]>0.4)
 then Maskedit3.Text := 'THINNER'

end;
```

DAFTAR PUSTAKA

[1] Johansson, Staffan, 2005, ”New Frequency Counting Pinciple Improves Resolution", Marketing Manager at Pendulum Instruments AB, Bromma, Sweden.

[2] Boven, Paul, 2007, ”Increasing The Resolution of Reciprocal Frequency Counters”, Tagung Weinheim.

[3] Fundamentals of the Electronic Counters,”Application Note 200 Electronic Counter Series”. Agilent Technologies. United States.

[4] Nakamoto, T., dan Moriizimi,T., 1998, ”Odor Sensor Using Quartz Resonator Array dan Neural Network Pattern Recognition”, Proceeding Ultrason.Symp.

[5] Jatmiko, Wisnu, dan Kusumoputro, Benyamin, 2012, ”Optimasi Vektor Codebooks Menggunakan Analisa Matriks Similaritas pada FLVQ”, Program Studi Ilmi Komputer Pasca Sarjana Fasilkom. Universitas Indonesia.

[6] Bystrom, Emil, “Construction of a Portable Piezoelectric Quartz Crystal Sensor Array for Determination of Petroleum Compounds in Contaminated Soil”, Umea Universitet, p.7.

[7] Anonim, 2014, ”QCM100-Quartz Crystal Microbalance Theory and Calibration”, Standford Research Systems.

[8] Doni Saldiro, 2009, ”Mengenal Komponen Dasar Kristal”, Majalah Elektronika Online.

[9] Jie, Han, 2006, ”Technical Background, Applications and Implementation of Quartz Crystal Microbalance Systems”, University of Jyvaskyla Department of Physics.

[10] Rivai, Muhammad, 2007,”Identifikasi Jenis Uap Pelarut Organik Berdasarkan Perubahan Frekuensi Resonansi Kristal SiO_2 Terlapis Polimer Menggunakan Jaringan Saraf Tiruan”, Universitas Airlangga.

[11] Lairan, Aldi, 2010, “Perancangan Sistem Pengenalan Jenis Odor Menggunakan 20Mhz *Quartz Crystal Microbalance* dan FPGA”, Laporan Tugas Akhir. Jurusan Teknik Elektro Institut Teknologi Sepuluh Nopember.

[12] Digital Electronics, 2014 ”Crystal Oscillators”, Faculty of Engineering and Information Technology. University of Technology Sydney.

[13] DeHon, Andre, dan Rubin, Raphael, 2004, "Design of FPGA Interconnect for Multilevel Metallization", IEEE Transactions on Very Large Scale Integration (VLSI) Systems.
[14] Lindenberg, F. Mayer, 2009, "High-Level FPGA Programming through Mapping Process Network to FPGA Resources", Institute of Computer Technology, Technical University of Hamburg-Harburg.
[15] Xilinx, 2013, "Spartan-3E FPGA Family: Introduction and Ordering Information", United States.
[16] Xilinx, 2011," Spartan-3 Generation FPGA User Guide, Extended Spartan-3A, Spartan-3E, and Spartan-3 FPGA Families", United States.
[17] Kusumadewi, Sri, 2004, "Membangun Jaringan Syaraf Tiruan Menggunakan MATLAB dan EXCEL-LINK", Graha Ilmu, Yogyakarta.
[18] Gerhenson, Carlos, 2003, "Artificial Neural Network for Beginners", United Kingdom.
[19] Haykin, S, 1999, "Neural Networks: a comprehensive fundation", Prentice Hall, New Jersey.
[20] He, Qin, 1999, "Neural Network and Its Application in IR", Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign.
[21] Ciptadi, Aditya, 2013," Pengembangan Sistem Hidung Elektronik Menggunakan Kominukasi Serial Bluetooth Pada PC Tablet", Laporan Tugas Akhir. Jurusan Teknik Elektro Institut Teknologi Sepuluh Nopember.

RIWAYAT HIDUP

Penulis dilahirkan di Surabaya pada tanggal 12 Oktober 1991. Sebagai anak terakhir dari empat bersaudara, penulis mengawali kegiatan pendidikan di SDN Pacarkeling VII Surabaya, dilanjutkan di SMP Negeri 9 Surabaya, SMA Negeri 16 Surabaya, hingga tahun 2009 diterima sebagai mahasiswa di jurusan Teknik Elektro ITS. Selama menjalani pendidikan di perguruan tinggi, penulis aktif dalam kepanitian baik itu di jurusan maupun di institut. Penulis juga berpartisipasi sebagai asisten laboratorium elektronika dasar.

Email : azura.rezo@gmail.com
barkah09@mhs.ee.its.ac.id